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: Акимов К.К.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 03.01.25

Москва, 2025

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

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.
- 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 дескриптором fd, в память, начиная с адреса start.
- int munmap(void *start, size_t length); – удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к 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); – использует указатель на динамическую библиотеку, возвращаемую dlopen, и оканчивающееся нулем символьное имя, а затем возвращает адрес, указывающий, откуда загружается этот символ. Если символ не найден, то возвращаемым значением dlsym является NULL;
- int dlclose(void *handle); – уменьшает на единицу счетчик ссылок на указатель динамической библиотеки.

Аллокатор памяти на основе списка свободных блоков

Это один из классических методов управления динамической памятью. Он основан на поддержании списка, в котором хранятся все доступные (свободные) участки памяти. Когда программе требуется выделить память, аллокатор ищет в этом списке подходящий свободный блок. После использования выделенную память возвращают обратно, и она снова попадает в список свободных блоков.

Основные компоненты и принципы работы:

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

Каждый элемент (блок) обычно содержит:

- Размер блока: количество байтов, доступных в этом свободном блоке.
- Указатель на следующий свободный блок: используется для связывания элементов списка.
- Возможны дополнительные поля, такие как флаг “свободен” или метаинформация для целей отладки.

Сама структура аллокатора обычно включает:

- Указатель на память; ● Указатель на голову списка; ● Размер памяти.

Операции:

1) Выделение памяти:

1. Аллокатор просматривает список свободных блоков, чтобы найти подходящий блок, размер которого больше или равен запрошенному.
2. Если подходящий блок найден, то происходит его “разрез”: часть блока выделяется для пользователя, а оставшаяся часть (если есть) остается в списке свободных блоков. Если размер подходящего блока точно соответствует запрошенному, он полностью удаляется из списка.
3. Возвращается указатель на выделенную область памяти.

2) Освобождение памяти:
   1. Аллокатор получает указатель на ранее выделенную область.
   2. Исходя из заголовка блока, он определяет размер освобождаемого блока.
   3. Освобождаемый блок возвращается в список свободных блоков.
   4. Также часто происходит слияние освобожденного блока со смежными свободными блоками для предотвращения фрагментации.

3) Инициализация:
   1. Выделяется начальный большой блок памяти.
   2. Этот блок добавляется в список свободных блоков.

4) Стратегии поиска свободного блока:
   1. First-Fit (первый подходящий): ищет первый блок, который достаточно большой для запроса. Простая реализация и высокая скорость поиска блока, но сильно увеличивается фрагментация.
   2. Best-Fit (наилучший подходящий): ищет наименьший блок, который достаточно большой для запроса. Ведет к меньшей фрагментации, но требует просмотра всего списка, что увеличивает время выделения памяти.

Преимущества:

- Простота реализации: основные алгоритмы достаточно просты для понимания и кодирования.
- Гибкость: может быть адаптирован к различным сценариям использования и требованиям.

Недостатки:

- Фрагментация: при частом выделении и освобождении блоков возникает фрагментация – множество мелких свободных блоков, непригодных для выделения под большие запросы.
- Зависимость от стратегии: эффективность сильно зависит от выбранной стратегии поиска свободного блока и сценария использования.
- Затраты на поиск: стратегия best-fit требует просмотра всего списка, что может быть

Аллокатор памяти на основе блоков 2^n

Этот тип аллокатора использует подход, основанный на выделении блоков памяти размером, являющимся степенью двойки. Он эффективно управляет выделением и освобождением памяти, минимизируя фрагментацию.

Основные компоненты и принципы работы:

Информация о свободных блоках хранится в массиве (m) списков свободных блоков, где индекс массива (i) удовлетворяет условию: m[i].size == min_block_size * 2^i
Каждый элемент (блок) обычно содержит:

- Размер блока: количество байтов, доступных в этом свободном блоке (степень 2).
- Указатель на следующий свободный блок такого же размера: используется для связывания элементов списка.
- Возможны дополнительные поля, такие как флаг “свободен” или метаинформация для целей отладки.

Сама структура аллокатора обычно включает:

- Указатель на память;
- Размер памяти;
- Массив списков (динамический или статический); ● Количество списков (если массив динамический); ● Минимальный размер блока.

Операции:

1) Выделение памяти:
    1. Аллокатор ищет блок памяти размером 2^n, который равен или больше запрошенного размера. Индекс массива определяется по степени двойки.
    2. Если по найденному индексу есть блок, то он удаляется из списка блоков, иначе начинается обход блоков больших размеров.
    3. Если найден блок размером в 2 и более раз больше нужного, он разбивается на более мелкие блоки, один из которых будет выделен под запрашиваемую память.
    4. Возвращается указатель на выделенную память.

2) Освобождение памяти:
    1. Аллокатор получает указатель на ранее выделенную область.
    2. Исходя из заголовка блока, он определяет размер освобождаемого блока и высчитывает по какому индексу нужно сделать вставку (есть альтернативный вариант, у занятых блоков хранить указатель на начало списка блоков такого же размера).
    3. Освобождаемый блок возвращается в список свободных блоков.

3) Инициализация (один из подходов):
    1. Инициализируется массив списков.
    2. Выделяется блок максимальной возможной степени 2, а остальные ячейки массива заполняются блоками оставшейся памяти.

Преимущества:

- Высокая скорость поиска: поиск блока определённого размера значительно упрощен, т.к. блоки разбиты по степеням двойки.
- Простая реализация: аллокатор имеет относительно простую структуру, что упрощает его разработку и отладку.
- Отсутствие внешней фрагментации: поскольку размеры блоков фиксированы (степени двойки), не возникает внешней фрагментации.
- Минимизация накладных расходов: отсутствие перераспределения и объединения блоков уменьшает накладные расходы на управление памятью.

Недостатки:

- Неэффективность для нестандартных запросов: если размер запроса не является степенью двойки, то аллокатор будет выделять блок памяти, большего размера, чем требуется.
- Внутренняя сегментация: оптимальный размер блока может не совпадать с реальными потребностями программы, что создаёт пустоты в памяти, которые нельзя использовать, до освобождения всего блока.
- Отсутствие слияния свободных блоков: может привести к отсутствию блоков большого размера после длительного использования одного аллокатора.

Тестирование

Для тестирования работы алгоритмов, было создано несколько тестов. С помощью них проверяется корректность работы аллокаторов в целом, а также определяются основные характеристики аллокаторов.

1) Тест на корректность выделения памяти: выделяем большой кусок памяти размером в половину доступной памяти, если он выделился, то пытаемся выделить блок размером больше половины памяти. Второй блок не должен выделиться, иначе аллокатор работает некорректно.
2) Тест на фрагментацию: выделяем много блоков памяти и сохраняем указатели в массив. Затем очищаем сначала чётные блоки, потом нечётные, после этого выделяем большой блок памяти. Данный тест позволяет проверить устойчивость аллокатора к фрагментации. Если большой блок в конце удастся выделить, то алгоритм имеет некоторую устойчивость к внешней сегментации.
3) Тест на переиспользование блоков: для того чтобы убедится в том, что блоки корректно освобождаются и могут быть переиспользованы, выделяем блок памяти, после освобождаем его, затем опять выделяем блок того же размера. Если указатели на освобожд ённый блок и на новый блок совпали, значит освобождение памяти работает корректно.
4) Тест на время выделения/освобождения памяти: замерим время выделения некоторого количества блоков памяти разных размеров, затем время освобождения этих же блоков. Для усложнения данного теста замерим время выделения и освобождения памяти в фрагментированном аллокаторе.
5) Тест на удобство использования: проверим удобство аллокаторов на базовой задаче. Например, создание динамической матрицы, заполнение значениями, вывод на экран и освобождение памяти.
6) Проверка фактора использования: в разных местах программы при разной загруженности аллокатора замеряем запрашиваемую и требуемую память. По этим данным вычисляем фактор использования

Результаты тестирования:

1) Оба аллокатора успешно прошли 1, 2 и 3 тесты, что говорит о корректности их работы.
2) Время освобождения и выделения памяти при малой сегментации у обоих аллокаторов примерно равно. Всё резко меняется при сегментированности данных в аллокаторе, при высокой сегментации списковый аллокатор становится намного медленнее, чем аллокатор 2^n. Это обусловлено линейным поиском блока для выделения памяти (при худшем случае), а также линейным поиском места для вставки освобождённого блока и слиянием. Аллокатор 2^n лишён данных недостатков, так как блоки одного размера хранятся по одному индексу, их легко выделять и освобождать.
3) Фактор использования был измерен при разных обстоятельствах:
    a) При выделении памяти под матрицу: списковый – 83.01%; 2^n – 52.63%;

b) При выделении большого количества блоков среднего размера: списковый – 91.53%; 2^n – 67.58%
c) При освобождении половины выделенных подряд блоков: списковый – 84.39%; 2^n – 63.60%

В указанных тестах списковый аллокатор показал немного лучшие результаты, это обусловлено тем, что он поддерживает слияние блоков и выделение блоков любого размера. Аллокатор 2^n использует большее количество памяти из-за отсутствия слияния блоков и фиксированных размеров выделяемых блоков (степень двойки).

4) Оба аллокатора удобно использовать в базовых задачах, не требующих большого количества выделений памяти за единицу времени, так как они работают через одинаковый API, близкий к стандартному

Код программы

Main.c

```c
#include "allocator.h"

#include <unistd.h>
#include <dlfcn.h>
#include <assert.h>
#include <string.h>
#include <time.h>

#define MEM_SIZE 9999999999

static size_t __USE_MEMORY = 0;

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


typedef struct fallback_allocator // резервный аллокатор + его функции
{
    size_t size;
    void *memory;
} fallback_allocator;

static Allocator *fallback_allocator_create(void *const memory, const size_t
size)
{
    if (memory == NULL || size <= sizeof(fallback_allocator))
    {
        return NULL;
    }
```

```
    fallback_allocator *alloc = (fallback_allocator *)((uint8_t *)memory);
    alloc->memory = memory + sizeof(fallback_allocator);
    alloc->size = size - sizeof(fallback_allocator);

    __USE_MEMORY = sizeof(fallback_allocator);

    return (Allocator *)alloc;
}

static void fallback_allocator_destroy(Allocator *const allocator)
{
    if (allocator == NULL)
        return;

    ((fallback_allocator *)allocator)->memory = NULL;
    ((fallback_allocator *)allocator)->size = 0;
}

// выделение памяти
static void *fallback_allocator_alloc(Allocator *const allocator, const size_t
size)
{
    if (allocator == NULL || size == 0)
    {
        return NULL;
    }

    static size_t offset = 0;

    fallback_allocator *all = (fallback_allocator *)allocator;
    if (offset + size > ((fallback_allocator *)allocator)->size)
    {
        return NULL;
    }

    void *allocated_memory = (void *)((char *)((fallback_allocator
*)allocator)->memory + offset);
    offset += size;

    __USE_MEMORY += size;

    return allocated_memory;
}

static void fallback_allocator_free(Allocator *const allocator, void *const
memory)
{ //освобождения памяти ( по факту ничего не делает)
    (void)allocator;
    (void)memory;
}
```

```c
static size_t fallback_allocator_get_used_memory()
{
    return __USE_MEMORY;
}


// тестикс
void test_edge_cases(Allocator *allocator, size_t total_size)
{
    // Тест на исчерпание памяти
    size_t alloc_size = total_size / 2;
    void *ptr1 = allocator_alloc(allocator, alloc_size);
    if (ptr1 == NULL)
    {
        char msg[] = "test_edge_cases: memory allocation error\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
    }

    void *ptr2 = allocator_alloc(allocator, alloc_size + 1);
    if (ptr2 != NULL)
    {
        char msg[] = "test_edge_cases: allocating more memory than the
available size\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        allocator_free(allocator, ptr2);
    }

    allocator_free(allocator, ptr1);

    {
        char msg[] = "test_edge_cases - OK\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        allocator_free(allocator, ptr2);
    }
}

void test_fragmentation_list(Allocator *allocator, size_t total_size)
{
    if (allocator == NULL)
        return;
    // Выделим и освободим несколько блоков в случайном порядке, чтобы создать
фрагментацию
    void *ptrs[1000];
    for (int i = 0; i < 1000; i++)
    {
        ptrs[i] = allocator_alloc(allocator, 100 + i * 10);
        if (ptrs[i] == NULL)
        {
            for (int j = 0; j < i; j++)
            {
                allocator_free(allocator, ptrs[i]);
```

```
            }
            char msg[] = "test_fragmentation_list - BAD\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
            return;
        }
    }
    for (int i = 0; i < 1000; i += 2)
    {
        allocator_free(allocator, ptrs[i]);
    }
    for (int i = 1; i < 1000; i += 2)
    {
        allocator_free(allocator, ptrs[i]);
    }

    void *large_ptr = allocator_alloc(allocator, total_size / 2);
    if (large_ptr == NULL)
    {
        char msg[] = "test_fragmentation_list - BAD\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        return;
    }

    allocator_free(allocator, large_ptr);
    char msg[] = "test_fragmentation_list - OK\n";
    write(STDOUT_FILENO, msg, sizeof(msg));
}

void test_reuse_blocks(Allocator *allocator, size_t total_size)
{ // используются ли освобожденные блоки снова?
    if (allocator == NULL)
        return;
    // Выделяем блок
    void *ptr1 = allocator_alloc(allocator, 128);
    if (ptr1 == NULL)
    {
        char msg[] = "test_reuse_blocks: error\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        return;
    }
    // Освобождаем блок
    allocator_free(allocator, ptr1);
    // Выделяем снова блок такого же размера
    void *ptr2 = allocator_alloc(allocator, 128);
    if (ptr2 == NULL)
    {
        char msg[] = "test_reuse_blocks: error\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        return;
    }
```

```
    // Проверяем, что блоки повторно используются
    if (ptr1 == ptr2)
    {
        char msg[] = "test_reuse_blocks - Reused same block +\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
    }
    else
    {
        char msg[] = "test_reuse_blocks - Reused diff block -\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
    }
    allocator_free(allocator, ptr2);
}

void test_alloc_and_free_time(Allocator *allocator)
{ // измерение времени при создании и освобождении 10 тыш блоков
    clock_t start, stop;
    double cpu_time_used;
    if (!allocator)
        return;

    void *ptrs[100000];

    start = clock();
    for (int i = 0; i < 100000; i++)
    {
        ptrs[i] = allocator_alloc(allocator, (10 + i) % 200 + 1);
        if (ptrs[i] == NULL)
        {
            char msg[] = "test_alloc_and_free_time: error\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
            return;
        }
    }
    stop = clock();
    cpu_time_used = ((double)(stop - start)) / CLOCKS_PER_SEC;
    {
        char msg[200];
        sprintf(msg, "alloc time: %f seconds\n", cpu_time_used);
        write(STDOUT_FILENO, msg, strlen(msg));
    }

    start = clock();
    for (int i = 0; i < 100000; i++)
    {
        allocator_free(allocator, ptrs[i]);
        if (ptrs[i] == NULL)
        {
            char msg[] = "test_alloc_and_free_time: error\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
            return;
```

```
        }
    }
    stop = clock();
    cpu_time_used = ((double)(stop - start)) / CLOCKS_PER_SEC;
    {
        char msg[200];
        sprintf(msg, "free time: %f seconds\n", cpu_time_used);
        write(STDOUT_FILENO, msg, strlen(msg));
    }

    for (int i = 0; i < 100000; i++)
    {
        ptrs[i] = allocator_alloc(allocator, (10 + i) % 100 + 1);
        if (ptrs[i] == NULL)
        {
            char msg[] = "test_alloc_and_free_time: error\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
            return;
        }
    }
    start = clock();
    for (int i = 0; i < 100000; i += 2)
    {
        allocator_free(allocator, ptrs[i]);
    }
    stop = clock();
    cpu_time_used = ((double)(stop - start)) / CLOCKS_PER_SEC;
    {
        char msg[200];
        sprintf(msg, "free time (with segmentation): %f seconds\n",
cpu_time_used);
        write(STDOUT_FILENO, msg, strlen(msg));
    }

    start = clock();
    for (int i = 0; i < 100000; i += 2)
    {
        ptrs[i] = allocator_alloc(allocator, (10 + i) % 300 + 1);
        if (ptrs[i] == NULL)
        {
            char msg[] = "test_alloc_and_free_time: error\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
            return;
        }
    }
    stop = clock();
    cpu_time_used = ((double)(stop - start)) / CLOCKS_PER_SEC;
    {
        char msg[200];
        sprintf(msg, "alloc time (with segmentation): %f seconds\n",
cpu_time_used);
```

```
        write(STDOUT_FILENO, msg, strlen(msg));
    }

    for (int i = 0; i < 100000; i++)
    {
        allocator_free(allocator, ptrs[i]);
    }
}



int main(int argc, char *argv[])
{
    size_t _MEM_USED = 0;
    void *library = NULL;

    if (argc == 1)
    {
        char msg[] = "The path to the library has not been transmitted,
enabling backup functions.\n";
        write(STDOUT_FILENO, msg, sizeof(msg));

        allocator_create = fallback_allocator_create;
        allocator_destroy = fallback_allocator_destroy;
        allocator_alloc = fallback_allocator_alloc;
        allocator_free = fallback_allocator_free;
        get_used_memory = fallback_allocator_get_used_memory;
    }
    else
    {
        library = dlopen(argv[1], RTLD_LOCAL | RTLD_NOW);
        {
            char msg[] = "Opening the library.\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
        }
        if (library)
        {
            {
                char msg[] = "Loading functions from the library.\n";
                write(STDOUT_FILENO, msg, sizeof(msg));
            }
            allocator_create = dlsym(library, "allocator_create");
            allocator_destroy = dlsym(library, "allocator_destroy");
            allocator_alloc = dlsym(library, "allocator_alloc");
            allocator_free = dlsym(library, "allocator_free");
            get_used_memory = dlsym(library, "get_used_memory");
        }
        if (!library || !allocator_create || !allocator_destroy ||
!allocator_alloc || !allocator_free || !get_used_memory)
        {
            {
```

```
                char msg[] = "Enabling backup functions.\n";
                write(STDOUT_FILENO, msg, sizeof(msg));
            }
            allocator_create = fallback_allocator_create;
            allocator_destroy = fallback_allocator_destroy;
            allocator_alloc = fallback_allocator_alloc;
            allocator_free = fallback_allocator_free;
            get_used_memory = fallback_allocator_get_used_memory;
        }
    }


    size_t memory_size = MEM_SIZE;
    void *memory = mmap(NULL, memory_size, PROT_READ | PROT_WRITE, MAP_PRIVATE
| MAP_ANONYMOUS, -1, 0);

    if (memory == MAP_FAILED)
    {
        char msg[] = "mmap failed\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        if (library)
            dlclose(library);
        return EXIT_FAILURE;
    }

    Allocator *allocator = allocator_create(memory, memory_size);

    if (allocator == NULL)
    {
        char msg[] = "allocator_create failed\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        munmap(memory, memory_size);
        if (library)
            dlclose(library);
        return EXIT_FAILURE;
    }

    // тесты
    {

        int rows = 20, cols = 20;
        int **matrix;
        int num = 0;

        // Выделение памяти под строки матрицы
        matrix = (int **)allocator_alloc(allocator, rows * sizeof(int *));

        _MEM_USED += rows * sizeof(int *);

        if (matrix == NULL)
        {
```

```
            char msg[] = "allocator_alloc failed\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
            munmap(memory, memory_size);
            if (library)
                dlclose(library);
            return EXIT_FAILURE;
        }

        // Выделение памяти под столбцы для каждой строки
        for (int i = 0; i < rows; i++)
        {
            matrix[i] = (int *)allocator_alloc(allocator, cols * sizeof(int));

            _MEM_USED += cols * sizeof(int);

            if (matrix[i] == NULL)
            {
                for (int j = 0; j < i; j++)
                {
                    allocator_free(allocator, matrix[j]);
                }
                allocator_free(allocator, matrix);
                {
                    char msg[] = "allocator_alloc failed\n";
                    write(STDOUT_FILENO, msg, sizeof(msg));
                    munmap(memory, memory_size);
                    if (library)
                        dlclose(library);
                    return EXIT_FAILURE;
                }
            }
        }

        {
            char msg[128]; // процент использования памяти ( лок. переменная /
фактически используемая память)
            sprintf(msg, "Factor (create dynamic matrix): %.3lf%%\n",
(double)_MEM_USED / get_used_memory() * 100.);
            write(STDOUT_FILENO, msg, strlen(msg));
        }
        // Заполнение матрицы
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                matrix[i][j] = num++;
            }
        }

        // Вывод матрицы
        for (int i = 0; i < rows; i++)
```

```
        {
            for (int j = 0; j < cols; j++)
            {
                char msg[10];
                sprintf(msg, "%4d ", matrix[i][j]);
                write(STDOUT_FILENO, msg, strlen(msg));
            }
            write(STDOUT_FILENO, "\n", 1);
        }

        // Освобождение памяти
        for (int i = 0; i < rows; i++)
        {
            allocator_free(allocator, matrix[i]);
            _MEM_USED -= cols * sizeof(int);
            matrix[i] = NULL;
        }
        allocator_free(allocator, matrix);
        _MEM_USED -= rows * sizeof(int *);
        matrix = NULL;
    }

    // Проверка фактора при сегментации
    {
        void *arr[100000];
        for (int i = 0; i < 100000; i++)
        {
            arr[i] = allocator_alloc(allocator, 173);

            _MEM_USED += 173;

            if (arr[i] == NULL)
            {
                for (int j = 0; j < i; j++)
                {
                    allocator_free(allocator, arr[j]);
                }
                {
                    char msg[] = "allocator_alloc failed\n";
                    write(STDOUT_FILENO, msg, sizeof(msg));
                    munmap(memory, memory_size);
                    if (library)
                        dlclose(library);
                    return EXIT_FAILURE;
                }
            }
        }
        {
            char msg[64];
            sprintf(msg, "Factor (many allocs): %.3lf%%\n", (double)_MEM_USED /
get_used_memory() * 100.);
```

```
            write(STDOUT_FILENO, msg, strlen(msg));
        }
        for (int i = 0; i < 100000; i += 2)
        {
            allocator_free(allocator, arr[i]);
            _MEM_USED -= 173;
        }
        {
            char msg[64];
            sprintf(msg, "Factor (1/2 allocs free): %.3lf%%\n",
(double)_MEM_USED / get_used_memory() * 100.);
            write(STDOUT_FILENO, msg, strlen(msg));
        }
        for (int i = 1; i < 100000; i += 2)
        {
            allocator_free(allocator, arr[i]);
            _MEM_USED -= 173;
        }
    }

    test_edge_cases(allocator, memory_size);
    test_fragmentation_list(allocator, memory_size);
    test_reuse_blocks(allocator, memory_size);
    test_alloc_and_free_time(allocator);

    allocator_destroy(allocator);

    if (munmap(memory, memory_size) == -1)
    {
        char msg[] = "munmap failed\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        if (library)
            dlclose(library);
        return EXIT_FAILURE;
    }
    if (library)
    {
        dlclose(library);
        library = NULL;
    }

    return 0;
}
```

Allocator_list.c:

```
#include "allocator.h"

#ifdef _MSC_VER
#define EXPORT __attribute__((visibility("default"))) // заменяем
```

```c
#else
#define EXPORT // просто убираем
#endif

#define __DEBUG 1

#if __DEBUG
static size_t __USED_MEMORY = 0;
#endif

// Структура свободного блока памяти
typedef struct block_header{
    size_t size;                // Размер блока в байтах
    struct block_header *next; // Указатель на следующий свободный блок
} block_header;

// Структура аллокатора
struct Allocator{
    block_header *free_list_head;
    void *memory;
    size_t size;
};

// Функция для инициализации аллокатора
EXPORT Allocator *allocator_create(void *const memory, const size_t size){
    if (memory == NULL || size <= sizeof(Allocator) + sizeof(block_header)){
        return NULL;
    }
    Allocator *allocator = (Allocator *)memory;
    allocator->memory = (uint8_t *)memory + sizeof(Allocator); // memory к типу
uint8_t
    allocator->size = size - sizeof(Allocator);

    allocator->free_list_head = (block_header *)allocator->memory; // память,
кроме заголовка
    allocator->free_list_head->size = allocator->size - sizeof(block_header);
// размер = все - размер заголовка
    allocator->free_list_head->next = NULL;

#if __DEBUG // проверяем, определен ли макрос
    __USED_MEMORY += sizeof(allocator) + sizeof(block_header);
#endif

    return allocator;
}

// Функция для деинициализации аллокатора
EXPORT void allocator_destroy(Allocator *const allocator){

    if (allocator == NULL){
        return;
```

```
    }
    allocator->free_list_head = NULL;
    allocator->memory = NULL;
    allocator->size = 0;

#if __DEBUG
    __USED_MEMORY -= sizeof(allocator) + sizeof(block_header);
#endif
}

// Функция для поиска свободного блока (Best-Fit)
static block_header *find_free_block(Allocator *const allocator, size_t size){

    if (allocator == NULL || allocator->free_list_head == NULL)
        return NULL;

    block_header *current = allocator->free_list_head;
    block_header *best_block = NULL;
    size_t minSize = __SIZE_MAX__;

    while (current != NULL){

        if (current->size >= size && current->size < minSize){

            best_block = current;
            minSize = current->size;
        }
        current = current->next;
    }
    return best_block;
}


// Функция для выделения памяти ( возвращает указатель)
EXPORT void *allocator_alloc(Allocator *const allocator, const size_t size) {
    if (allocator == NULL || size <= 0) {
        return NULL;
    }

    block_header *free_block = allocator->free_list_head; // начало списка
блоков

    // Ищем первый подходящий блок
    while (free_block != NULL && free_block->size < size) {
        free_block = free_block->next;
    }

    if (free_block == NULL) {
        return NULL; // Нет подходящего блока
    }
```

```
    // место = указатель на норм блок + размер структуры блока
    void *allocated_address = (uint8_t *)free_block + sizeof(block_header);

    // Если блок слишком большой, разделим его (берем нужный размер, остальное
как новый блок записываем в список блоков)
    if (free_block->size > size) {
        block_header *new_free_block = (block_header *)((uint8_t *)free_block +
sizeof(block_header) + size);
        new_free_block->size = free_block->size - size - sizeof(block_header);
        new_free_block->next = free_block->next;
        free_block->next = new_free_block;
        free_block->size = size;

        if (free_block == allocator->free_list_head) {
            allocator->free_list_head = new_free_block;
        } else {
            block_header *current = allocator->free_list_head;
            while (current != NULL && current->next != free_block) {
                current = current->next;
            }
            if (current != NULL) {
                current->next = new_free_block;
            }
        }

#if __DEBUG
        __USED_MEMORY += sizeof(block_header);
#endif
    } else {
        // Удаляем текущий блок из списка свободных (типа берем весь)
        if (free_block == allocator->free_list_head) {
            allocator->free_list_head = free_block->next;
        } else {
            block_header *current = allocator->free_list_head;
            while (current != NULL && current->next != free_block) {
                current = current->next;
            }
            if (current != NULL) {
                current->next = free_block->next;
            }
        }
    }

#if __DEBUG
    __USED_MEMORY += size;
#endif

    return allocated_address;
}
```

```
// Слияние соседних свободных блоков
static void merge_blocks(Allocator *allocator){ //обходим список и соединяем
небольшие блоки в один
    block_header *prev, *current;
    if (allocator == NULL)
        return; // Некорректный аллокатор или пустой список

    prev = allocator->free_list_head;
    if (prev == NULL)
        return;

    current = prev->next;
    if (current == NULL)
        return;

    while (current != NULL){
        // проверка что блоки соседи по памяти
        if ((uint8_t *)prev + sizeof(block_header) + prev->size == (uint8_t
*)current){
            prev->size += current->size + sizeof(block_header);
            prev->next = current->next;
            current = prev->next;

#if __DEBUG
            __USED_MEMORY -= sizeof(block_header);
#endif
        }
        else{
            prev = current;
            current = current->next;
        }
    }
}

// Функция для освобождения памяти
EXPORT void allocator_free(Allocator *const allocator, void *const memory){ //
адрес где надо освободить
    if (allocator == NULL || memory == NULL)
        return;

    block_header *new_free_block = (block_header *)((char *)memory -
sizeof(block_header));

    // Ищем куда вставить освободившийся блок в список
    // начало ( если пусто)
    if (allocator->free_list_head == NULL || (uint8_t *)new_free_block <
(uint8_t *)allocator->free_list_head){
        new_free_block->next = allocator->free_list_head;
```

```
        allocator->free_list_head = new_free_block;
    }
    else{
        // ищем место в списке ( по порядку) типо если наш после текущего - 
вставляем, иначе идем дальше
        block_header *current = allocator->free_list_head;
        block_header *prev = NULL;
        while (current != NULL && (uint8_t *)new_free_block > (uint8_t 
*)current){
            prev = current;
            current = current->next;
        }
        new_free_block->next = current;
        if (prev != NULL){
            prev->next = new_free_block;
        }
    }

#if __DEBUG
    __USED_MEMORY -= new_free_block->size;
#endif

    merge_blocks(allocator);
    block_header *tmp = allocator->free_list_head;
}

#if __DEBUG
EXPORT size_t get_used_memory(){
    return __USED_MEMORY;
}
#endif
```

Allocator_2n.c:

```
#include "allocator.h"

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

static size_t __USED_MEMORY = 0;

// Структура для представления блока
typedef struct block_header{
    size_t size;               // Размер блока (включая этот заголовок)
```

```
    struct block_header *next; // Указатель на следующий свободный блок в списке
} block_header;

// Структура для управления аллокатором
typedef struct Allocator
{
    void *mem_start;           // Начало области памяти mmap
    size_t mem_size;           // Размер области mmap
    block_header **free_lists; // Массив списков свободных блоков
    size_t num_lists;          // Количество списков
    size_t min_block_size;     // Минимальный размер блока (например, 8 байт)
} Allocator;

// Функция для вычисления индекса списка для заданного размера
static size_t get_list_index(Allocator const *allocator, size_t size)
{
    size_t block_size = allocator->min_block_size;
    size_t index = 0;
    while (block_size < size)
    {
        block_size <<= 1;
        index++;
    }
    return index;
}

static size_t align_to_power_of_two(size_t size)
{
    size_t current_size = 2;
    while (current_size < size)
    {
        current_size <<= 1;
    }
    return current_size;
}

// Функция для инициализации аллокатора
EXPORT Allocator *allocator_create(void *const memory, const size_t size)
{
    if (size < 512 || memory == NULL)
        return NULL;

    size_t min_block_size = align_to_power_of_two(sizeof(block_header) + 4);
    // Вычисляем количество списков
    size_t num_lists = 0;
    size_t block_size_temp = min_block_size;
    while (block_size_temp <= size)
    {
        num_lists++;
        block_size_temp <<= 1; // Умножаем на 2
    }

    // Выделяем память под структуру аллокатора и под списки
    Allocator *allocator = (Allocator *)memory;
    size_t lists_size = num_lists * sizeof(block_header *);
```

```
    allocator->free_lists = (block_header **)((uint8_t *)memory + sizeof(Allocator));
// списки после аллокатора

    allocator->mem_start = memory;
    allocator->mem_size = size;
    allocator->num_lists = num_lists;
    allocator->min_block_size = min_block_size;

    // Инициализируем списки свободных блоков
    for (size_t i = 0; i < num_lists; i++)
    {
        allocator->free_lists[i] = NULL;
    }

    // Заполняем массивы free_lists блоками, начиная с наибольшего
    size_t usable_memory_start = sizeof(Allocator) + lists_size;
    size_t current_block_start = usable_memory_start;
    size_t mem_remains = size - usable_memory_start; // Инициализируем блок размером
оставшейся памяти

    for (int i = num_lists - 1; i >= 0; i--)
    {
        if (mem_remains < min_block_size)
            break;

        size_t target_block_size = min_block_size << i;
        if (current_block_start < size && target_block_size <= mem_remains)
        {

            block_header *block = (block_header *)((uint8_t *)memory +
current_block_start);
            __USED_MEMORY += sizeof(block_header);
            block->size = target_block_size;
            block->next = NULL;

            allocator->free_lists[i] = block;         // Помещаем блок в нужный список
            current_block_start += target_block_size; // Обновляем позицию следующего
блока
            mem_remains -= target_block_size;         // Уменьшаем размер доступной
памяти
        }
    }
    __USED_MEMORY += usable_memory_start;

    return allocator;
}

// Функция для выделения памяти
EXPORT void *allocator_alloc(Allocator *const allocator, const size_t _size)
{
    if (allocator == NULL)
        return NULL;

    // Учитываем заголовок блока при выделении
    size_t size = align_to_power_of_two(_size + sizeof(block_header));
```

```
    size_t index = get_list_index(allocator, size);

    if (index >= allocator->num_lists)
        return NULL;

    block_header *block = NULL;
    if (allocator->free_lists[index])
    {
        block = allocator->free_lists[index];
        allocator->free_lists[index] = allocator->free_lists[index]->next;
        __USED_MEMORY += size - sizeof(block_header);
        return (void *)((uint8_t *)block + sizeof(block_header));
    }

    // Если не нашли блок нужного размера, ищем в списках с большими блоками
    for (size_t i = index + 1; i < allocator->num_lists; i++)
    {
        if (allocator->free_lists[i] != NULL)
        {
            block = allocator->free_lists[i];
            allocator->free_lists[i] = block->next;
            // Разбиваем блок
            while (block->size > size)
            {
                size_t split_size = block->size >> 1; // Делим блок пополам

                if (split_size < size)
                {
                    break; // Нельзя разбить блок на блок нужного размера
                }

                block->size = split_size;

                block_header *new_block = (block_header *)((uint8_t *)block +
split_size);

                __USED_MEMORY += sizeof(block_header);

                new_block->size = split_size;
                // Помещаем новый блок в соответствующий список
                size_t new_block_index = get_list_index(allocator, split_size);
                new_block->next = allocator->free_lists[new_block_index];
                allocator->free_lists[new_block_index] = new_block;
            }
            break;
        }
    }

    if (block == NULL)
        return NULL;

    __USED_MEMORY += size - sizeof(block_header);
    return (void *)((uint8_t *)block + sizeof(block_header));
}
```

```c
EXPORT void allocator_free(Allocator *const allocator, void *const memory)
{
    if (allocator == NULL || memory == NULL)
        return;

    block_header *block = (block_header *)((uint8_t *)memory - sizeof(block_header));

    size_t index = get_list_index(allocator, block->size);
    if (index >= allocator->num_lists)
        return;

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

    __USED_MEMORY -= block->size - sizeof(block_header);
}

EXPORT void allocator_destroy(Allocator *const allocator)
{
    if (allocator == NULL)
        return;

    allocator->free_lists = NULL;
    allocator->mem_size = 0;
    allocator->mem_start = NULL;
    allocator->min_block_size = 0;
    allocator->num_lists = 0;
}

EXPORT size_t get_used_memory()
{
    return __USED_MEMORY;
}
```

Alloactor.h:

```c
#ifndef OSI_ALLOCATOR_H
#define OSI_ALLOCATOR_H

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/mman.h>
#include <unistd.h>
#include <errno.h>
#include <limits.h>
#include <stdint.h>

typedef struct Allocator Allocator;

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);
typedef size_t get_used_memory_f();


#endif
```

Протокол работы программы

kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab4/src$ ./main ./allocator_list.so

Opening the library.

Loading functions from the library.

Factor (create dynamic matrix): 83.019%

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399

Factor (many allocs): 91.534%

Factor (1/2 allocs free): 84.390%

test_edge_cases - OK

test_fragmentation_list - OK

test_reuse_blocks - Reused same block + alloc time: 0.001128

seconds free time: 0.001687 seconds

free time (with segmentation): 7.087616 seconds

alloc time (with segmentation): 8.035285 seconds

kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab4/src$ ./main ./allocator_2n.so

Opening the library.

Loading functions from the library.

Factor (create dynamic matrix): 52.632%

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399

Factor (many allocs): 67.576%

Factor (1/2 allocs free): 63.599%

test_edge_cases - OK

test_fragmentation_list - OK

test_reuse_blocks - Reused same block + alloc time: 0.006752

seconds free time: 0.007160 seconds

free time (with segmentation): 0.004901 seconds

alloc time (with segmentation): 0.005230 seconds

kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab4/src$ ./main ./library.so

Opening the library.

Enabling backup functions.

Factor (create dynamic matrix): 99.099%

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105

106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130
131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155
156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205
206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230
231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280
281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330
331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355
356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380
381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399

Factor (many allocs): 99.990%

Factor (1/2 allocs free): 49.995%

test_edge_cases - OK

test_fragmentation_list - BAD

test_reuse_blocks - Reused diff block - alloc time: 0.000567

seconds free time: 0.000358 seconds

free time (with segmentation): 0.000193 seconds

alloc time (with segmentation): 0.000410 seconds

kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab4/src$ strace ./main ./allocator_2n.so

execve("./main", ["./main", "./allocator_2n.so"], 0x7ffdaf00e7a8 /* 28 vars */) = 0 brk(NULL) = 0x5593ac903000 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f5191a81000 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=20231, ...}) = 0 mmap(NULL, 20231, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f5191a7c000 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) = 0x7f519186a000 mmap(0x7f5191892000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f5191892000 mmap(0x7f5191a1a000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f5191a1a000 mmap(0x7f5191a69000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f5191a69000 mmap(0x7f5191a6f000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f5191a6f000 close(3) = 0 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f5191867000 arch_prctl(ARCH_SET_FS, 0x7f5191867740) = 0 set_tid_address(0x7f5191867a10) = 6482 set_robust_list(0x7f5191867a20, 24) = 0 rseq(0x7f5191868060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f5191a69000, 16384, PROT_READ) = 0 mprotect(0x5593abf0b000, 4096, PROT_READ) = 0 mprotect(0x7f5191ab9000, 8192, PROT_READ) = 0 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=81921024, rlim_max=RLIM64_INFINITY}) = 0 munmap(0x7f5191a7c000, 20231) = 0 getrandom("\x7c\x56\x9a\x48\x98\x93\xcb\x84", 8, GRND_NONBLOCK) = 8 brk(NULL) = 0x5593ac903000 brk(0x5593ac924000) = 0x5593ac924000

openat(AT_FDCWD, "./allocator_2n.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|0777, st_size=15408, ...}) = 0

getcwd("/mnt/c/Users/User/OSI/lab4/src", 128) = 31

mmap(NULL, 16408, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f5191a7c000

mmap(0x7f5191a7d000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f5191a7d000 mmap(0x7f5191a7e000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f5191a7e000 mmap(0x7f5191a7f000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f5191a7f000 close(3) = 0 mprotect(0x7f5191a7f000, 4096, PROT_READ) = 0 write(1, "Opening the library.\n\0", 22Opening the library.) = 22 write(1, "Loading functions from the libra"..., 37Loading functions from the library.) = 37

mmap(NULL, 9999999999, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f4f3d7a8000

write(1, "Factor (create dynamic matrix): "..., 40Factor (create dynamic matrix): 52.632%) = 40 write(1, " 0 ", 5 0) = 5 write(1, " 1 ", 5 1) = 5 write(1, " 2 ", 5 2) = 5 write(1, " 3 ", 5 3) = 5 write(1, " 4 ", 5 4) = 5 write(1, " 5 ", 5 5) = 5 write(1, " 6 ", 5 6) = 5 write(1, " 7 ", 5 7) = 5 write(1, " 8 ", 5 8) = 5 write(1, " 9 ", 5 9) = 5 write(1, " 10 ", 5 10) = 5 write(1, " 11 ", 5 11) = 5 write(1, " 12 ", 5 12) = 5 write(1, " 13 ", 5 13) = 5 write(1, " 14 ", 5 14) = 5 write(1, " 15 ", 5 15) = 5 write(1, " 16 ", 5 16) = 5 write(1, " 17 ", 5 17) = 5 write(1, " 18 ", 5 18) = 5 write(1, " 19 ", 5 19) = 5 write(1, "\n", 1) = 1 write(1, " 20 ", 5 20) = 5 write(1, " 21 ", 5 21) = 5 write(1, " 22 ", 5 22) = 5 write(1, " 23 ", 5 23) = 5 write(1, " 24 ", 5 24) = 5 write(1, " 25 ", 5 25) = 5 write(1, " 26 ", 5 26) = 5 write(1, " 27 ", 5 27) = 5 write(1, " 28 ", 5 28) = 5 write(1, " 29 ", 5 29) = 5 write(1, " 30 ", 5 30) = 5 write(1, " 31 ", 5 31) = 5 write(1, " 32 ", 5 32) = 5 write(1, " 33 ", 5 33) = 5 write(1, " 34 ", 5 34) = 5 write(1, " 35 ", 5 35) = 5 write(1, " 36 ", 5 36) = 5 write(1, " 37 ", 5 37) = 5 write(1, " 38 ", 5 38) = 5 write(1, " 39 ", 5 39) = 5 write(1, "\n", 1) = 1 write(1, " 40 ", 5 40) = 5 write(1, " 41 ", 5 41) = 5 write(1, " 42 ", 5 42) = 5 write(1, " 43 ", 5 43) = 5 write(1, " 44 ", 5 44) = 5 write(1, " 45 ", 5 45) = 5 write(1, " 46 ", 5 46) = 5 write(1, " 47 ", 5 47) = 5 write(1, " 48 ", 5 48) = 5 write(1, " 49 ", 5 49) = 5 write(1, " 50 ", 5 50) = 5 write(1, " 51 ", 5 51) = 5 write(1, " 52 ", 5 52) = 5 write(1, " 53 ", 5 53) = 5 write(1, " 54 ", 5 54) = 5 write(1, " 55 ", 5 55) = 5 write(1, " 56 ", 5 56) = 5 write(1, " 57 ", 5 57) = 5 write(1, " 58 ", 5 58) = 5 write(1, " 59 ", 5 59) = 5 write(1, "\n", 1) = 1 write(1, " 60 ", 5 60) = 5 write(1, " 61 ", 5 61) = 5 write(1, " 62 ", 5 62) = 5 write(1, " 63 ", 5 63) = 5 write(1, " 64 ", 5 64) = 5 write(1, " 65 ", 5 65) = 5 write(1, " 66 ", 5 66) = 5 write(1, " 67 ", 5 67) = 5 write(1, " 68 ", 5 68) = 5 write(1, " 69 ", 5 69) = 5 write(1, " 70 ", 5 70) = 5 write(1, " 71 ", 5 71) = 5 write(1, " 72 ", 5 72) = 5 write(1, " 73 ", 5 73) = 5 write(1, " 74 ", 5 74) = 5 write(1, " 75 ", 5 75) = 5 write(1, " 76 ", 5 76) = 5 write(1, " 77 ", 5 77) = 5 write(1, " 78 ", 5 78) = 5 write(1, " 79 ", 5 79) = 5 write(1, "\n", 1) = 1 write(1, " 80 ", 5 80) = 5 write(1, " 81 ", 5 81) = 5 write(1, " 82 ", 5 82) = 5 write(1, " 83 ", 5 83) = 5 write(1, " 84 ", 5 84) = 5 write(1, " 85 ", 5 85) = 5 write(1, " 86 ", 5 86) = 5 write(1, " 87 ", 5 87) = 5 write(1, " 88 ", 5 88) = 5 write(1, " 89 ", 5 89) = 5 write(1, " 90 ", 5 90) = 5 write(1, " 91 ", 5 91) = 5 write(1, " 92 ", 5 92) = 5 write(1, " 93 ", 5 93) = 5 write(1, " 94 ", 5 94) = 5 write(1, " 95 ", 5 95) = 5 write(1, " 96 ", 5 96) = 5 write(1, " 97 ", 5 97) = 5 write(1, " 98 ", 5 98) = 5 write(1, " 99 ", 5 99) = 5 write(1, "\n", 1) = 1 write(1, " 100 ", 5 100) = 5 write(1, " 101 ", 5 101) = 5 write(1, " 102 ", 5 102) = 5 write(1, " 103 ", 5 103) = 5 write(1, " 104 ", 5 104) = 5 write(1, " 105 ", 5 105) = 5 write(1, " 106 ", 5 106) = 5 write(1, " 107 ", 5 107) = 5 write(1, " 108 ", 5 108)

= 5 write(1, " 109 ", 5 109) = 5 write(1, " 110 ", 5 110) = 5 write(1, " 111 ", 5 111) = 5 write(1, " 112 ", 5 112) = 5 write(1, " 113 ", 5 113) = 5 write(1, " 114 ", 5 114) = 5 write(1, " 115 ", 5 115) = 5 write(1, " 116 ", 5 116) = 5 write(1, " 117 ", 5 117) = 5 write(1, " 118 ", 5 118) = 5 write(1, " 119 ", 5 119) = 5 write(1, "\n", 1) = 1 write(1, " 120 ", 5 120) = 5 write(1, " 121 ", 5 121) = 5 write(1, " 122 ", 5 122) = 5 write(1, " 123 ", 5 123) = 5 write(1, " 124 ", 5 124) = 5 write(1, " 125 ", 5 125) = 5 write(1, " 126 ", 5 126) = 5 write(1, " 127 ", 5 127) = 5 write(1, " 128 ", 5 128) = 5 write(1, " 129 ", 5 129) = 5 write(1, " 130 ", 5 130) = 5 write(1, " 131 ", 5 131) = 5 write(1, " 132 ", 5 132) = 5 write(1, " 133 ", 5 133) = 5 write(1, " 134 ", 5 134) = 5 write(1, " 135 ", 5 135) = 5 write(1, " 136 ", 5 136) = 5 write(1, " 137 ", 5 137) = 5 write(1, " 138 ", 5 138) = 5 write(1, " 139 ", 5 139) = 5 write(1, "\n", 1) = 1 write(1, " 140 ", 5 140) = 5 write(1, " 141 ", 5 141) = 5 write(1, " 142 ", 5 142) = 5 write(1, " 143 ", 5 143) = 5 write(1, " 144 ", 5 144) = 5 write(1, " 145 ", 5 145) = 5 write(1, " 146 ", 5 146) = 5 write(1, " 147 ", 5 147) = 5 write(1, " 148 ", 5 148) = 5 write(1, " 149 ", 5 149) = 5 write(1, " 150 ", 5 150) = 5 write(1, " 151 ", 5 151) = 5 write(1, " 152 ", 5 152) = 5 write(1, " 153 ", 5 153) = 5 write(1, " 154 ", 5 154) = 5 write(1, " 155 ", 5 155) = 5 write(1, " 156 ", 5 156) = 5 write(1, " 157 ", 5 157) = 5 write(1, " 158 ", 5 158) = 5 write(1, " 159 ", 5 159) = 5 write(1, "\n", 1) = 1 write(1, " 160 ", 5 160) = 5 write(1, " 161 ", 5 161) = 5 write(1, " 162 ", 5 162) = 5 write(1, " 163 ", 5 163) = 5 write(1, " 164 ", 5 164) = 5 write(1, " 165 ", 5 165) = 5 write(1, " 166 ", 5 166) = 5 write(1, " 167 ", 5 167) = 5 write(1, " 168 ", 5 168) = 5 write(1, " 169 ", 5 169) = 5 write(1, " 170 ", 5 170) = 5 write(1, " 171 ", 5 171) = 5 write(1, " 172 ", 5 172) = 5 write(1, " 173 ", 5 173) = 5 write(1, " 174 ", 5 174) = 5 write(1, " 175 ", 5 175) = 5 write(1, " 176 ", 5 176) = 5 write(1, " 177 ", 5 177) = 5 write(1, " 178 ", 5 178) = 5 write(1, " 179 ", 5 179) = 5 write(1, "\n", 1) = 1 write(1, " 180 ", 5 180) = 5 write(1, " 181 ", 5 181) = 5 write(1, " 182 ", 5 182) = 5 write(1, " 183 ", 5 183) = 5 write(1, " 184 ", 5 184) = 5 write(1, " 185 ", 5 185) = 5 write(1, " 186 ", 5 186) = 5 write(1, " 187 ", 5 187) = 5 write(1, " 188 ", 5 188) = 5 write(1, " 189 ", 5 189) = 5 write(1, " 190 ", 5 190) = 5 write(1, " 191 ", 5 191) = 5 write(1, " 192 ", 5 192) = 5 write(1, " 193 ", 5 193) = 5 write(1, " 194 ", 5 194) = 5 write(1, " 195 ", 5 195) = 5 write(1, " 196 ", 5 196) = 5 write(1, " 197 ", 5 197) = 5 write(1, " 198 ", 5 198) = 5 write(1, " 199 ", 5 199) = 5 write(1, "\n", 1) = 1 write(1, " 200 ", 5 200) = 5 write(1, " 201 ", 5 201) = 5 write(1, " 202 ", 5 202) = 5 write(1, " 203 ", 5 203) = 5 write(1, " 204 ", 5 204) = 5 write(1, " 205 ", 5 205) = 5 write(1, " 206 ", 5 206) = 5 write(1, " 207 ", 5 207) = 5 write(1, " 208 ", 5 208) = 5 write(1, " 209 ", 5 209) = 5 write(1, " 210 ", 5 210) = 5 write(1, " 211 ", 5 211) = 5 write(1, " 212 ", 5 212) = 5 write(1, " 213 ", 5 213) = 5 write(1, " 214 ", 5 214) = 5 write(1, " 215 ", 5 215) = 5 write(1, " 216 ", 5 216) = 5 write(1, " 217 ", 5 217) = 5 write(1, " 218 ", 5 218) = 5 write(1, " 219 ", 5 219) = 5 write(1, "\n", 1) = 1 write(1, " 220 ", 5 220) = 5 write(1, " 221 ", 5 221) = 5 write(1, " 222 ", 5 222) = 5 write(1, " 223 ", 5 223) = 5 write(1, " 224 ", 5 224) = 5 write(1, " 225 ", 5 225) = 5 write(1, " 226 ", 5 226) = 5 write(1, " 227 ", 5 227) = 5 write(1, " 228 ", 5 228) = 5 write(1, " 229 ", 5 229) = 5 write(1, " 230 ", 5 230) = 5 write(1, " 231 ", 5 231) = 5 write(1, " 232 ", 5 232) = 5 write(1, " 233 ", 5 233) = 5 write(1, " 234 ", 5 234) = 5 write(1, " 235 ", 5 235) = 5 write(1, " 236 ", 5 236) = 5 write(1, " 237 ", 5 237) = 5 write(1, " 238 ", 5 238) = 5 write(1, " 239 ", 5 239) = 5 write(1, "\n", 1) = 1 write(1, " 240 ", 5 240) = 5 write(1, " 241 ", 5 241) = 5 write(1, " 242 ", 5 242) = 5 write(1, " 243 ", 5 243) = 5 write(1, " 244 ", 5 244) = 5 write(1, " 245 ", 5 245) = 5 write(1, " 246 ", 5 246) = 5 write(1, " 247 ", 5 247) = 5 write(1, " 248 ", 5 248) = 5 write(1, " 249 ", 5 249) = 5 write(1, " 250 ", 5 250) = 5 write(1, " 251 ", 5 251) = 5 write(1, " 252 ", 5 252) = 5 write(1, " 253 ", 5 253) = 5 write(1, " 254 ", 5 254) = 5 write(1, " 255 ", 5 255) = 5 write(1, " 256 ", 5 256) = 5 write(1, " 257 ", 5 257) = 5 write(1, " 258 ", 5 258) = 5 write(1, " 259 ", 5 259) = 5 write(1, "\n", 1) = 1 write(1, " 260 ", 5 260) = 5 write(1, " 261 ", 5 261) = 5 write(1, " 262 ", 5 262) = 5 write(1, " 263 ", 5 263) = 5 write(1, " 264 ", 5 264) = 5 write(1, " 265 ", 5 265) = 5 write(1, " 266 ", 5 266) = 5 write(1, " 267 ", 5 267) = 5 write(1, " 268 ", 5 268) = 5 write(1, " 269 ", 5 269) = 5 write(1, " 270 ", 5 270) = 5 write(1, " 271 ", 5 271) = 5 write(1, " 272 ", 5 272) = 5 write(1, " 273 ", 5 273) = 5 write(1, " 274 ", 5 274) = 5 write(1, " 275 ", 5 275) = 5 write(1, " 276 ", 5 276) = 5 write(1, " 277 ", 5 277) = 5 write(1, " 278 ", 5 278) = 5 write(1, " 279 ", 5 279) = 5 write(1, "\n", 1) = 1 write(1, " 280 ", 5 280) = 5 write(1, " 281 ", 5 281) = 5 write(1, " 282 ", 5 282) = 5 write(1, " 283 ", 5 283) = 5 write(1, " 284 ", 5 284) = 5 write(1, " 285 ", 5 285) = 5 write(1, " 286 ", 5 286) = 5 write(1, " 287 ", 5 287) = 5 write(1, " 288 ", 5 288) = 5 write(1, " 289 ", 5 289) = 5 write(1, " 290 ", 5 290) = 5 write(1, " 291 ", 5 291) = 5 write(1, " 292 ", 5 292) = 5 write(1, " 293 ", 5 293) = 5 write(1, " 294 ", 5 294) = 5 write(1, " 295 ", 5 295) = 5 write(1, " 296 ", 5 296) = 5 write(1, " 297 ", 5 297) = 5 write(1, " 298 ", 5 298) = 5 write(1, " 299 ", 5 299) = 5 write(1, "\n", 1) = 1 write(1, " 300 ", 5 300) = 5 write(1, " 301 ", 5 301) = 5 write(1, " 302 ", 5 302) = 5 write(1, " 303 ", 5 303) = 5 write(1, " 304 ", 5 304) = 5 write(1, " 305 ", 5 305) = 5

write(1, " 306 ", 5 306) = 5 write(1, " 307 ", 5 307) = 5 write(1, " 308 ", 5 308) = 5 write(1, " 309 ", 5 309) = 5 write(1, " 310 ", 5 310) = 5 write(1, " 311 ", 5 311) = 5 write(1, " 312 ", 5 312) = 5 write(1, " 313 ", 5 313) = 5 write(1, " 314 ", 5 314) = 5 write(1, " 315 ", 5 315) = 5 write(1, " 316 ", 5 316) = 5 write(1, " 317 ", 5 317) = 5 write(1, " 318 ", 5 318) = 5 write(1, " 319 ", 5 319) = 5 write(1, "\n", 1) = 1 write(1, " 320 ", 5 320) = 5 write(1, " 321 ", 5 321) = 5 write(1, " 322 ", 5 322) = 5 write(1, " 323 ", 5 323) = 5 write(1, " 324 ", 5 324) = 5 write(1, " 325 ", 5 325) = 5 write(1, " 326 ", 5 326) = 5 write(1, " 327 ", 5 327) = 5 write(1, " 328 ", 5 328) = 5 write(1, " 329 ", 5 329) = 5 write(1, " 330 ", 5 330) = 5 write(1, " 331 ", 5 331) = 5 write(1, " 332 ", 5 332) = 5 write(1, " 333 ", 5 333) = 5 write(1, " 334 ", 5 334) = 5 write(1, " 335 ", 5 335) = 5 write(1, " 336 ", 5 336) = 5 write(1, " 337 ", 5 337) = 5 write(1, " 338 ", 5 338) = 5 write(1, " 339 ", 5 339) = 5 write(1, "\n", 1) = 1 write(1, " 340 ", 5 340) = 5 write(1, " 341 ", 5 341) = 5 write(1, " 342 ", 5 342) = 5 write(1, " 343 ", 5 343) = 5 write(1, " 344 ", 5 344) = 5 write(1, " 345 ", 5 345) = 5 write(1, " 346 ", 5 346) = 5 write(1, " 347 ", 5 347) = 5 write(1, " 348 ", 5 348) = 5 write(1, " 349 ", 5 349) = 5 write(1, " 350 ", 5 350) = 5 write(1, " 351 ", 5 351) = 5 write(1, " 352 ", 5 352) = 5 write(1, " 353 ", 5 353) = 5 write(1, " 354 ", 5 354) = 5 write(1, " 355 ", 5 355) = 5 write(1, " 356 ", 5 356) = 5 write(1, " 357 ", 5 357) = 5 write(1, " 358 ", 5 358) = 5 write(1, " 359 ", 5 359) = 5 write(1, "\n", 1) = 1 write(1, " 360 ", 5 360) = 5 write(1, " 361 ", 5 361) = 5 write(1, " 362 ", 5 362) = 5 write(1, " 363 ", 5 363) = 5 write(1, " 364 ", 5 364) = 5 write(1, " 365 ", 5 365) = 5 write(1, " 366 ", 5 366) = 5 write(1, " 367 ", 5 367) = 5 write(1, " 368 ", 5 368) = 5 write(1, " 369 ", 5 369) = 5 write(1, " 370 ", 5 370) = 5 write(1, " 371 ", 5 371) = 5 write(1, " 372 ", 5 372) = 5 write(1, " 373 ", 5 373) = 5 write(1, " 374 ", 5 374) = 5 write(1, " 375 ", 5 375) = 5 write(1, " 376 ", 5 376) = 5 write(1, " 377 ", 5 377) = 5 write(1, " 378 ", 5 378) = 5 write(1, " 379 ", 5 379) = 5 write(1, "\n", 1) = 1 write(1, " 380 ", 5 380) = 5 write(1, " 381 ", 5 381) = 5 write(1, " 382 ", 5 382) = 5 write(1, " 383 ", 5 383) = 5 write(1, " 384 ", 5 384) = 5 write(1, " 385 ", 5 385) = 5 write(1, " 386 ", 5 386) = 5 write(1, " 387 ", 5 387) = 5 write(1, " 388 ", 5 388) = 5 write(1, " 389 ", 5 389) = 5 write(1, " 390 ", 5 390) = 5 write(1, " 391 ", 5 391) = 5 write(1, " 392 ", 5 392) = 5 write(1, " 393 ", 5 393) = 5 write(1, " 394 ", 5 394) = 5 write(1, " 395 ", 5 395) = 5 write(1, " 396 ", 5 396) = 5 write(1, " 397 ", 5 397) = 5 write(1, " 398 ", 5 398) = 5 write(1, " 399 ", 5 399) = 5 write(1, "\n", 1) = 1

write(1, "Factor (many allocs): 67.576%\n", 30Factor (many allocs): 67.576%) = 30 write(1, "Factor (1/2 allocs free): 63.599"..., 34Factor (1/2 allocs free): 63.599%) = 34 write(1, "test_edge_cases - OK\n\0", 22test_edge_cases - OK) = 22 write(1, "test_fragmentation_list - OK\n\0", 30test_fragmentation_list - OK) = 30 write(1, "test_reuse_blocks - Reused same "..., 41test_reuse_blocks - Reused same block +) = 41
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=83490100}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=87174000}) = 0 write(1, "alloc time: 0.003684 seconds\n", 29alloc time: 0.003684 seconds) = 29
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=87400800}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=91577500}) = 0 write(1, "free time: 0.004177 seconds\n", 28free time: 0.004177 seconds) = 28
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=96126000}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=97728000}) = 0 write(1, "free time (with segmentation): 0"..., 48free time (with segmentation): 0.001602 seconds) = 48
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=97910000}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=100624700}) = 0 write(1, "alloc time (with segmentation): "..., 49alloc time (with segmentation): 0.002714 seconds) = 49

munmap(0x7f4f3d7a8000, 9999999999) = 0

munmap(0x7f5191a7c000, 16408) = 0

exit_group(0) = ? +++ exited with 0 +++

kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab4/src$

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. Была создана резервная реализация аллокаторов через mmap. Работа аллокаторов была проверена на собственных тестах, а также проведена сравнительная характеристика двух алгоритмов. Проблем во время написания лабораторной работы не возникло.