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: Савков И.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 08.01.25

Москва, 2025

Постановка задачи

Вариант 8.

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 (см. пример на GitHub Gist). 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);

Необходимо реализовать:

8. Списки свободных блоков (наиболее подходящее) и алгоритм Мак-Кьюзика-Кэрелса;

Общий метод и алгоритм решения

Использованные системные вызовы:

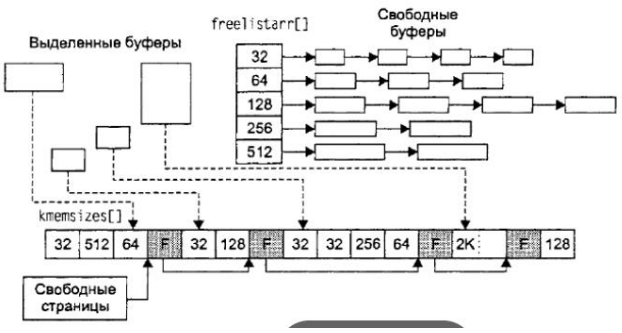
- mmap() – для выделения блока памяти (аналог malloc), используется для резервирования памяти для аллокатора.
- munmap() – для освобождения ранее выделенной памяти (аналог free), применяется для очистки ресурсов, выделенных через mmap().
- dlopen() – для загрузки динамической библиотеки во время выполнения программы, используется для подключения аллокатора из внешней библиотеки.
- dlsym() – для получения указателей на функции из динамически загруженной библиотеки, позволяет использовать функции аллокатора.
- dlclose() – для закрытия загруженной динамической библиотеки, освобождает ресурсы, связанные с загрузкой библиотеки.

Реализованы аллокаторы:

Аллокатор McKusick-Karels: Является улучшенной версией алгоритма «Блоки по 2n»

- Храним массив страниц
  - Страница может быть свободной и хранить ссылку на следующую свободную страницу
  - Может быть разбита на блоки - в массиве содержится размер блока, на которые разбита страница
  - Указатель на то, что страница является частью крупного блока памяти
- Храним массив с указателями на списки свободных блоков
  - Блоки имеют размер равный степени 2
  - Блоки внутри одной страницы имеют одинаковый размер

Илюстрация



Аллокатор памяти на основе списка свободных блоков

Основные принципы:

1. **Список свободных блоков**:
   - Представляет собой структуру данных (обычно односвязный или двусвязный список).
   - Каждый элемент описывает свободный участок памяти.
2. **Структура блока**:
   - **Размер блока**: количество доступных байтов.
   - **Указатель на следующий свободный блок**: связывает элементы списка.

- Возможны дополнительные поля, такие как флаг “свободен” или метаинформация.

3. **Структура аллокатора**:
   - Указатель на память.
   - Указатель на голову списка свободных блоков.
   - Размер памяти.

Сравнение аллокаторов: алгоритма Мак-Кьюзика-Кэрелса и Списков свободных блоков

1. Фактор использования памяти

Фактор использования памяти оценивает эффективность использования доступной памяти и уровень фрагментации.

Алгоритм Мак-Кьюзика-Кэрелса

- Основан на "слоящихся списках" (slab allocation), где память разбивается на фиксированные блоки определенных размеров.
- Минимальная внешняя фрагментация благодаря фиксированным размерам блоков.
- Эффективен для приложений с предсказуемыми размерами запросов.
- Возможна перерасход памяти, если размеры запросов плохо совпадают с заранее настроенными блоками.

Списки свободных блоков

- Память разделена на произвольные блоки, которые организованы в связный список.
- При частом выделении и освобождении блоков возникают проблемы с фрагментацией (особенно при использовании **First-Fit**).
- Более универсален, так как не требует предсказания размеров запросов.

Итог:

- Алгоритм Мак-Кьюзика-Кэрелса обеспечивает более эффективное использование памяти при предсказуемых нагрузках.
- Списки свободных блоков могут страдать от большей фрагментации, особенно при нерегулярных запросах.

---

2. Скорость выделения блоков

Алгоритм Мак-Кьюзика-Кэрелса

- Выделение блоков из заранее подготовленных списков занимает константное время O(1)O(1).
- При необходимости выделения новой страницы или переполнении списка затраты возрастают.
- Хорошо справляется с частыми повторяющимися запросами типичных размеров.

Списки свободных блоков

- Скорость зависит от стратегии поиска:
  - **First-Fit** выполняется быстро, но может потребовать линейного обхода списка.
  - **Best-Fit** требует полного сканирования, что увеличивает время выполнения.
- Возможна оптимизация через индексацию или дополнительные структуры данных.

Итог:

- Алгоритм Мак-Кьюзика-Кэрелса быстрее при типичных размерах запросов.
- Списки свободных блоков могут быть медленнее, особенно при использовании ресурсоёмких стратегий поиска.

---

3. Скорость освобождения блоков

Алгоритм Мак-Кьюзика-Кэрелса

- Освобождение занимает O(1)O(1), так как блок возвращается в соответствующий список.
- Дополнительные затраты могут возникнуть при возврате страниц памяти обратно системе.

Списки свободных блоков

- Освобождение также выполняется быстро (O(1)O(1)), так как блоки возвращаются в начало списка.
- Важна корректная обработка слияния освобожденного блока с соседними, чтобы предотвратить фрагментацию.

Итог:

- Оба подхода имеют схожую скорость освобождения.

---

4. Простота реализации и использования

Алгоритм Мак-Кьюзика-Кэрелса

- Требует ручной настройки размеров блоков и анализа типичных запросов.
- Реализация сложнее из-за необходимости учитывать заранее определенные размеры.

Списки свободных блоков

- Просты в реализации и не требуют сложной настройки.
- Универсальны, подходят для приложений с разнообразными размерами запросов.

Итог:

- Списки свободных блоков проще в реализации и универсальнее в использовании.

Сравнение производительности аллокаторов

Тесты скорости:

Тест	Аллокатор Мак-Кьюзика-Кэрелса (ns)	Списки свободных блоков (ns)	Быстрее
Test 1 (Allocate)	151	141	Списки свободных блоков
Test 1 (Free)	40	70	Мак-Кьюзик-Кэрелс

Тест	Аллокатор Мак-Кьюзика-Кэрелса (ns)	Списки свободных блоков (ns)	Быстрее
Test 2 (Allocate)	40	70	Мак-Кьюзик-Кэрелс
Test 2 (Free)	60	80	Мак-Кьюзик-Кэрелс
Test 3 (Allocate)	90	100	Мак-Кьюзик-Кэрелс
Test 3 (Free)	40	130	Мак-Кьюзик-Кэрелс
Test 4 (Allocate)	50	90	Мак-Кьюзик-Кэрелс
Test 4 (Free)	40	81	Мак-Кьюзик-Кэрелс

Выводы:

1. Аллокатор Мак-Кьюзика-Кэрелса показывает лучшую производительность при частых малых запросах.
2. Списки свободных блоков уступают в скорости при сложных операциях освобождения и выделения.

Итоговое сравнение:

- **Аллокатор Мак-Кьюзика-Кэрелса** предпочтителен для систем с предсказуемыми нагрузками и частыми маломасштабными запросами.
- **Списки свободных блоков** универсальны, но их производительность и использование памяти сильно зависят от стратегии поиска и сценариев использования.

Код программы

allocator_mkk.c

```
#include <stddef.h>

#define ALIGN_SIZE(size, alignment) (((size) + (alignment - 1)) & ~(alignment - 1)) // Выравнивание размера
#define FREE_LIST_ALIGNMENT 8

typedef struct Block {
   struct Block *next_block;
} Block;

typedef struct Allocator {
   void *base_addr;
   size_t total_size;
   Block *free_list_head;
} Allocator;

Allocator *allocator_create(void *memory_region, size_t region_size);

void allocator_destroy(Allocator *allocator);

void *allocator_alloc(Allocator *allocator, size_t alloc_size);

void allocator_free(Allocator *allocator, void *memory_block);

Allocator *allocator_create(void *memory_region, const size_t region_size) {
```

```
    if (memory_region == NULL || region_size < sizeof(Allocator)) {
        return NULL;
    }

    Allocator *allocator = (Allocator *) memory_region;
    allocator->base_addr = (char *) memory_region + sizeof(Allocator);
    allocator->total_size = region_size - sizeof(Allocator);
    allocator->free_list_head = (Block *) allocator->base_addr;

    // Инициализация списка
    if (allocator->free_list_head != NULL) {
        allocator->free_list_head->next_block = NULL;
    }

    return allocator;
}


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

    size_t aligned_size = ALIGN_SIZE(alloc_size, FREE_LIST_ALIGNMENT);
    Block *previous_block = NULL;
    Block *current_block = allocator->free_list_head;

    while (current_block != NULL) {
        if (aligned_size <= allocator->total_size) { // достаточно ли места
            if (previous_block != NULL) {
                previous_block->next_block = current_block->next_block;
            } else {
                allocator->free_list_head = current_block->next_block;
            }
            return current_block; // возвращаем найденный блок
        }

        previous_block = current_block;
        current_block = current_block->next_block;
    }

    return NULL;
}

void allocator_free(Allocator *allocator, void *memory_block) {
    if (allocator == NULL || memory_block == NULL) {
        return;
    }

    Block *block_to_free = (Block *) memory_block;
    block_to_free->next_block = allocator->free_list_head;
    allocator->free_list_head = block_to_free; // Добавление блока обратно в список свободных блоков
}

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

    allocator->base_addr = NULL;
    allocator->total_size = 0;
    allocator->free_list_head = NULL;
}
```

allocator_freelist.c

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


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

typedef struct Allocator {
   Block *head;
   void *memory;
   size_t size;
} Allocator;

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

   Allocator *allocator = (Allocator *) mmap(NULL, sizeof(Allocator), PROT_READ | PROT_WRITE,
                                             MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   if (!allocator) return NULL;

   allocator->memory = memory;
   allocator->size = size;
   allocator->head = (Block *) memory;

   allocator->head->size = size - sizeof(Block);
   allocator->head->next = NULL;

   return allocator;
}

void allocator_destroy(Allocator *const allocator) {
   if (allocator) {
      munmap(allocator, sizeof(Allocator));
   }
}

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

   Block *current = allocator->head;
   Block *prev = NULL;
   Block *best_fit = NULL;
   Block *best_fit_prev = NULL;

   size_t best_fit_size = ULONG_MAX;

   // Найти лучший блок
   while (current) {
      if (current->size >= size && current->size < best_fit_size) {
         best_fit = current;
         best_fit_prev = prev;
         best_fit_size = current->size;
      }
      prev = current;
      current = current->next;
   }

   if (!best_fit) return NULL;
```

```
    size_t remaining_size = best_fit->size - size - sizeof(Block);

    if (remaining_size >= sizeof(Block)) {
        Block *new_block = (Block *) ((char *) best_fit + sizeof(Block) + size);
        new_block->size = remaining_size;
        new_block->next = best_fit->next;

        if (best_fit_prev) {
            best_fit_prev->next = new_block;
        } else {
            allocator->head = new_block;
        }
        best_fit->size = size;
    } else {
        if (best_fit_prev) {
            best_fit_prev->next = best_fit->next;
        } else {
            allocator->head = best_fit->next;
        }
    }

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

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

    Block *header = (Block *) ((char *) memory - sizeof(Block));
    Block *current = allocator->head;
    Block *prev = NULL;

    // Найти позицию для вставки
    while (current && current < header) {
        prev = current;
        current = current->next;
    }

    header->next = current;

    if (prev) {
        prev->next = header;
        if ((char *) prev + sizeof(Block) + prev->size == (char *) header) {
            prev->size += sizeof(Block) + header->size;
            prev->next = header->next;
            header = prev;
        }
    } else {
        allocator->head = header;
    }

    if (current && (char *) header + sizeof(Block) + header->size == (char *) current) {
        header->size += sizeof(Block) + current->size;
        header->next = current->next;
    }
}
```

main.c

```
/*
gcc -shared -fPIC -o libmckusick_carels.so mckusick_carels.c
gcc -shared -fPIC -o libblock_2n.so block_2n.c
gcc -o main main.c -ldl
```

```
./main ./libblock_2n.so
./main ./libmckusick_carels.so
*/

#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <unistd.h>
#include <string.h>
#include <sys/mman.h>
#include <time.h>
#define SNPRINTF_BUF 256

typedef void *(*allocator_create_t)(void *const memory, const size_t size);

typedef void *(*allocator_alloc_t)(void *const allocator, const size_t size);

typedef void (*allocator_free_t)(void *const allocator, void *const memory);

typedef void (*allocator_destroy_t)(void *const allocator);

typedef struct {
   char first_name[52];
   char last_name[52];
   char group[15];
} Student;

void write_message(const char *message) {
   write(STDOUT_FILENO, message, strlen(message));
}

void write_error(const char *message) {
   write(STDERR_FILENO, message, strlen(message));
}

long long calculate_time_diff_ns(const struct timespec start, const struct timespec end) {
   return (end.tv_sec - start.tv_sec) * 1000000000LL + (end.tv_nsec - start.tv_nsec);
}

int main(int argc, char *argv[]) {
   if (argc < 2) {
      write_error("Usage: <program> <path_to_allocator_library>\n");
      return 52;
   }

   void *allocator_lib = dlopen(argv[1], RTLD_LAZY);
   if (allocator_lib == NULL) {
      write_error("Error loading library: ");
      write_error(dlerror());
      write_error("\n");
      return 1;
   }

   allocator_create_t allocator_create = (allocator_create_t) dlsym(allocator_lib, "allocator_create");
   allocator_alloc_t allocator_alloc = (allocator_alloc_t) dlsym(allocator_lib, "allocator_alloc");
   allocator_free_t allocator_free = (allocator_free_t) dlsym(allocator_lib, "allocator_free");
   allocator_destroy_t allocator_destroy = (allocator_destroy_t) dlsym(allocator_lib, "allocator_destroy");

   if (!allocator_create || !allocator_alloc || !allocator_free || !allocator_destroy) {
      write_error("Error locating functions: ");
      write_error(dlerror());
      write_error("\n");
      dlclose(allocator_lib);
```

```
    return 1;
}

size_t pool_size = 4 * 1024 * 1024; // 4 MB
void *memory_pool = mmap(NULL, pool_size, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
if (memory_pool == MAP_FAILED) {
    write_error("Memory allocation for pool failed (mmap)\n");
    dlclose(allocator_lib);
    return 1;
}
write_message("POOL SIZE is 4 MB = 4 * 1024 * 1024\n");

void *allocator = allocator_create(memory_pool, pool_size);
if (!allocator) {
    write_error("Allocator creation failed\n");
    munmap(memory_pool, pool_size);
    dlclose(allocator_lib);
    return 1;
}

struct timespec start, end;

write_message("\n===============TEST1===============\n\n");
clock_gettime(CLOCK_MONOTONIC, &start);
Student *stud_array = (Student *) allocator_alloc(allocator, 5 * sizeof(Student));
clock_gettime(CLOCK_MONOTONIC, &end);

if (stud_array) {
    char buffer[SNPRINTF_BUF];
    snprintf(buffer, sizeof(buffer), "[[LOG]]Time to allocate: %lld ns\n", calculate_time_diff_ns(start, end));
    write_message(buffer);

    for (int i = 0; i < 5; i++) {
        snprintf(stud_array[i].first_name, sizeof(stud_array[i].first_name), "Vladimir%d", i + 1);
        snprintf(stud_array[i].last_name, sizeof(stud_array[i].last_name), "Bugrenkov%d", i + 1);
        snprintf(stud_array[i].group, sizeof(stud_array[i].group), "M8O-21%d-23", i);
    }

    write_message("Students before shuffle:\n");
    for (int i = 0; i < 5; i++) {
        snprintf(buffer, sizeof(buffer), "Name: %s, Last Name: %s, Group: %s\n",
                stud_array[i].first_name, stud_array[i].last_name, stud_array[i].group);
        write_message(buffer);
    }

    Student temp = stud_array[0];
    stud_array[0] = stud_array[4];
    stud_array[4] = temp;

    write_message("Students after shuffle:\n");
    for (int i = 0; i < 5; i++) {
        snprintf(buffer, sizeof(buffer), "Name: %s, Last Name: %s, Group: %s\n",
                stud_array[i].first_name, stud_array[i].last_name, stud_array[i].group);
        write_message(buffer);
    }

    clock_gettime(CLOCK_MONOTONIC, &start);
    allocator_free(allocator, stud_array);
    clock_gettime(CLOCK_MONOTONIC, &end);
    snprintf(buffer, sizeof(buffer), "[[LOG]]Time to free block: %lld ns\n", calculate_time_diff_ns(start, end));
    write_message(buffer);
} else {
    write_error("Memory allocation for students failed\n");
}
```

```
write_message("\n===============TEST2===============\n\n");
clock_gettime(CLOCK_MONOTONIC, &start);
long double *long_double_array = (long double *) allocator_alloc(allocator, 20 * sizeof(long double));
clock_gettime(CLOCK_MONOTONIC, &end);

if (long_double_array) {
   char buffer[SNPRINTF_BUF];
   snprintf(buffer, sizeof(buffer), "[[LOG]]Time to allocate: %lld ns\n", calculate_time_diff_ns(start, end));
   write_message(buffer);

   for (int i = 0; i < 20; i++) {
      long_double_array[i] = i + 1;
   }
   write_message("array before mulp52:\n");
   for (int i = 0; i < 20; i++) {
      char buffer[SNPRINTF_BUF];
      snprintf(buffer, sizeof(buffer), "long_double_array[%d] = %Lf\n", i, long_double_array[i]);
      write_message(buffer);
   }
   for (int i = 0; i < 20; i++) {
      long_double_array[i] *= 52;
   }
   write_message("array after mulp52:\n");
   for (int i = 0; i < 20; i++) {
      char buffer[SNPRINTF_BUF];
      snprintf(buffer, sizeof(buffer), "long_double_array[%d] = %Lf\n", i, long_double_array[i]);
      write_message(buffer);
   }
   clock_gettime(CLOCK_MONOTONIC, &start);
   allocator_free(allocator, long_double_array);
   clock_gettime(CLOCK_MONOTONIC, &end);
   snprintf(buffer, sizeof(buffer), "[[LOG]]Time to free block: %lld ns\n", calculate_time_diff_ns(start, end));
   write_message(buffer);
} else {
   write_error("Memory allocation for long double array failed\n");
}
write_message("\n===============TEST3===============\n\n");
clock_gettime(CLOCK_MONOTONIC, &start);
long double *large_array = (long double *) allocator_alloc(allocator, 5001 * sizeof(long double));
clock_gettime(CLOCK_MONOTONIC, &end);


if (large_array) {
   char buffer[SNPRINTF_BUF];
   snprintf(buffer, sizeof(buffer), "[[LOG]]Time to allocate: %lld ns\n", calculate_time_diff_ns(start, end));
   write_message(buffer);
   large_array[5000] = 52;
   snprintf(buffer, sizeof(buffer), "large_array[5000] = %Lf\n", large_array[5000]);
   write_message(buffer);
   write_message("Large array (5001 long double) allocated and freed successfully\n");

   clock_gettime(CLOCK_MONOTONIC, &start);
   allocator_free(allocator, large_array);
   clock_gettime(CLOCK_MONOTONIC, &end);
   snprintf(buffer, sizeof(buffer), "[[LOG]]Time to free block: %lld ns\n", calculate_time_diff_ns(start, end));
   write_message(buffer);
} else {
   write_error("Memory allocation for large array failed\n");
}

write_message("\n===============TEST4===============\n\n");
clock_gettime(CLOCK_MONOTONIC, &start);
```

```
    void *block1 = allocator_alloc(allocator, 3 * 1024 * 1024);

    clock_gettime(CLOCK_MONOTONIC, &end);

    if (block1) {
        char buffer[SNPRINTF_BUF];
        snprintf(buffer, sizeof(buffer), "[[LOG]]Time to allocate: %lld ns\n", calculate_time_diff_ns(start, end));
        write_message(buffer);

        write_message("Memory allocated (3 * 1024 * 1024 bytes)\n");
        clock_gettime(CLOCK_MONOTONIC, &start);
        allocator_free(allocator, block1);
        clock_gettime(CLOCK_MONOTONIC, &end);
        snprintf(buffer, sizeof(buffer), "[[LOG]]Time to free block: %lld ns\n", calculate_time_diff_ns(start, end));
        write_message(buffer);
        write_message("Memory freed (3 * 1024 * 1024 bytes)\n");
    } else {
        write_error("Memory allocation failed\n");
    }

    allocator_destroy(allocator);
    write_message("Allocator destroyed\n");
    munmap(memory_pool, pool_size);
    dlclose(allocator_lib);
    return 0;
}
```

Протокол работы программы

goldglaid@GoldGlaid0:/mnt/c/Users/GoldGlaid/CLionProjects/OSLab/lab4$ ./main ./liballocator_mkk.so 1024

POOL SIZE is 4 MB = 4 * 1024 * 1024

==========TEST1==========

[LOG]Time to allocate: 151 ns

Students before shuffle:

Name: Savkov1, Last Name: Igor1, Group: M8O-210-23

Name: Savkov2, Last Name: Igor2, Group: M8O-211-23

Name: Savkov3, Last Name: Igor3, Group: M8O-212-23

Name: Savkov4, Last Name: Igor4, Group: M8O-213-23

Name: Savkov5, Last Name: Igor5, Group: M8O-214-23

Students after shuffle:

Name: Savkov5, Last Name: Igor5, Group: M8O-214-23

Name: Savkov2, Last Name: Igor2, Group: M8O-211-23

Name: Savkov3, Last Name: Igor3, Group: M8O-212-23

Name: Savkov4, Last Name: Igor4, Group: M8O-213-23

Name: Savkov1, Last Name: Igor1, Group: M8O-210-23

[LOG]Time to free block: 40 ns

==========TEST2==========

[LOG]Time to allocate: 40 ns

array before mulp52:

long_double_array[0] = 1.000000

long_double_array[1] = 2.000000

long_double_array[2] = 3.000000

long_double_array[3] = 4.000000

long_double_array[4] = 5.000000

long_double_array[5] = 6.000000

long_double_array[6] = 7.000000

long_double_array[7] = 8.000000

long_double_array[8] = 9.000000

long_double_array[9] = 10.000000

long_double_array[10] = 11.000000

long_double_array[11] = 12.000000

long_double_array[12] = 13.000000

long_double_array[13] = 14.000000

long_double_array[14] = 15.000000

long_double_array[15] = 16.000000

long_double_array[16] = 17.000000

long_double_array[17] = 18.000000

long_double_array[18] = 19.000000

long_double_array[19] = 20.000000

array after mulp52:

long_double_array[0] = 52.000000

long_double_array[1] = 104.000000

long_double_array[2] = 156.000000

long_double_array[3] = 208.000000

long_double_array[4] = 260.000000

long_double_array[5] = 312.000000

long_double_array[6] = 364.000000

long_double_array[7] = 416.000000

long_double_array[8] = 468.000000

long_double_array[9] = 520.000000

long_double_array[10] = 572.000000

long_double_array[11] = 624.000000

long_double_array[12] = 676.000000

long_double_array[13] = 728.000000

long_double_array[14] = 780.000000

long_double_array[15] = 832.000000

long_double_array[16] = 884.000000

long_double_array[17] = 936.000000

long_double_array[18] = 988.000000

long_double_array[19] = 1040.000000

[LOG]Time to free block: 60 ns

==========TEST3==========

[LOG]Time to allocate: 90 ns

large_array[5000] = 52.000000

Large array (5001 long double) allocated and freed successfully

[LOG]Time to free block: 40 ns

==========TEST4==========

[LOG]Time to allocate: 50 ns

Memory allocated (3 * 1024 * 1024 bytes)

[LOG]Time to free block: 40 ns

Memory freed (3 * 1024 * 1024 bytes)

Allocator destroyed

goldglaid@GoldGlaid0:/mnt/c/Users/GoldGlaid/CLionProjects/OSLab/lab4$ ./main ./liballocator_freelist.so 1024

POOL SIZE is 4 MB = 4 * 1024 * 1024

==========TEST1==========

[LOG]Time to allocate: 141 ns

Students before shuffle:

Name: Savkov1, Last Name: Igor1, Group: M8O-210-23

Name: Savkov2, Last Name: Igor2, Group: M8O-211-23

Name: Savkov3, Last Name: Igor3, Group: M8O-212-23

Name: Savkov4, Last Name: Igor4, Group: M8O-213-23

Name: Savkov5, Last Name: Igor5, Group: M8O-214-23

Students after shuffle:

Name: Savkov5, Last Name: Igor5, Group: M8O-214-23

Name: Savkov2, Last Name: Igor2, Group: M8O-211-23

Name: Savkov3, Last Name: Igor3, Group: M8O-212-23

Name: Savkov4, Last Name: Igor4, Group: M8O-213-23

Name: Savkov1, Last Name: Igor1, Group: M8O-210-23

[LOG]Time to free block: 70 ns

==========TEST2==========

[LOG]Time to allocate: 70 ns

array before mulp52:

long_double_array[0] = 1.000000

long_double_array[1] = 2.000000

long_double_array[2] = 3.000000

long_double_array[3] = 4.000000

long_double_array[4] = 5.000000

long_double_array[5] = 6.000000

long_double_array[6] = 7.000000

long_double_array[7] = 8.000000

long_double_array[8] = 9.000000

long_double_array[9] = 10.000000

long_double_array[10] = 11.000000

long_double_array[11] = 12.000000

long_double_array[12] = 13.000000

long_double_array[13] = 14.000000

long_double_array[14] = 15.000000

long_double_array[15] = 16.000000

long_double_array[16] = 17.000000

long_double_array[17] = 18.000000

long_double_array[18] = 19.000000

long_double_array[19] = 20.000000

array after mulp52:

long_double_array[0] = 52.000000

long_double_array[1] = 104.000000

long_double_array[2] = 156.000000

long_double_array[3] = 208.000000

long_double_array[4] = 260.000000

long_double_array[5] = 312.000000

long_double_array[6] = 364.000000

long_double_array[7] = 416.000000

long_double_array[8] = 468.000000

long_double_array[9] = 520.000000

long_double_array[10] = 572.000000

long_double_array[11] = 624.000000

long_double_array[12] = 676.000000

long_double_array[13] = 728.000000

long_double_array[14] = 780.000000

long_double_array[15] = 832.000000

long_double_array[16] = 884.000000

long_double_array[17] = 936.000000

long_double_array[18] = 988.000000

long_double_array[19] = 1040.000000

[LOG]Time to free block: 80 ns

==========TEST3==========

[LOG]Time to allocate: 100 ns

large_array[5000] = 52.000000

Large array (5001 long double) allocated and freed successfully

[LOG]Time to free block: 130 ns

==========TEST4==========

[LOG]Time to allocate: 90 ns

Memory allocated (3 * 1024 * 1024 bytes)

[LOG]Time to free block: 81 ns

Memory freed (3 * 1024 * 1024 bytes)

Allocator destroyed

Strace:

```
goldglaid@GoldGlaid0:/mnt/c/Users/GoldGlaid/CLionProjects/OSLab/lab4$ strace ./main
./liballocator_mkk.so 1024
execve("./main", ["./main", "./liballocator_mkk.so", "1024"], 0x7ffde806d900 /* 26 vars */) = 0
brk(NULL)                               = 0x55d89ce64000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffcceaffe80) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fb3ea5bd000
```

```
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=25563, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 25563, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fb3ea5b6000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fb3ea38d000
mprotect(0x7fb3ea3b5000, 2023424, PROT_NONE) = 0
mmap(0x7fb3ea3b5000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fb3ea3b5000
mmap(0x7fb3ea54a000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7fb3ea54a000
mmap(0x7fb3ea5a3000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7fb3ea5a3000
mmap(0x7fb3ea5a9000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fb3ea5a9000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fb3ea38a000
arch_prctl(ARCH_SET_FS, 0x7fb3ea38a740) = 0
set_tid_address(0x7fb3ea38aa10)         = 75563
set_robust_list(0x7fb3ea38aa20, 24)     = 0
rseq(0x7fb3ea38b0e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7fb3ea5a3000, 16384, PROT_READ) = 0
mprotect(0x55d87a47d000, 4096, PROT_READ) = 0
mprotect(0x7fb3ea5f7000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fb3ea5b6000, 25563)           = 0
getrandom("\x18\x4d\x00\x85\x22\x2b\x4d\x65", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x55d89ce64000
brk(0x55d89ce85000)                     = 0x55d89ce85000
openat(AT_FDCWD, "./liballocator_mkk.so", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0777, st_size=15272, ...}, AT_EMPTY_PATH) = 0
getcwd("/mnt/c/Users/GoldGlaid/CLionProjects/OSLab/lab4", 128) = 48
mmap(NULL, 16424, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fb3ea5b8000
mmap(0x7fb3ea5b9000, 4096, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7fb3ea5b9000
mmap(0x7fb3ea5ba000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x2000) = 0x7fb3ea5ba000
mmap(0x7fb3ea5bb000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7fb3ea5bb000
close(3)                                = 0
mprotect(0x7fb3ea5bb000, 4096, PROT_READ) = 0
```

mmap(NULL, 4194304, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb3e9f8a000
write(1, "POOL SIZE is 4 MB = 4 * 1024 * 1"..., 36POOL SIZE is 4 MB = 4 * 1024 * 1024
) = 36
write(1, "\n==============TEST1==========="..., 38
==============TEST1==============

) = 38
write(1, "[LOG]Time to allocate: 1924 ns\n", 31[LOG]Time to allocate: 1924 ns
) = 31
write(1, "Students before shuffle:\n", 25Students before shuffle:
) = 25
write(1, "Name: Savkov1, Last Name: Igor1,"..., 53Name: Savkov1, Last Name: Igor1, Group: M8O-210-23
) = 53
write(1, "Name: Savkov2, Last Name: Igor2,"..., 53Name: Savkov2, Last Name: Igor2, Group: M8O-211-23
) = 53
write(1, "Name: Savkov3, Last Name: Igor3,"..., 53Name: Savkov3, Last Name: Igor3, Group: M8O-212-23
) = 53
write(1, "Name: Savkov4, Last Name: Igor4,"..., 53Name: Savkov4, Last Name: Igor4, Group: M8O-213-23
) = 53
write(1, "Name: Savkov5, Last Name: Igor5,"..., 53Name: Savkov5, Last Name: Igor5, Group: M8O-214-23
) = 53
write(1, "Students after shuffle:\n", 24Students after shuffle:
) = 24
write(1, "Name: Savkov5, Last Name: Igor5,"..., 53Name: Savkov5, Last Name: Igor5, Group: M8O-214-23
) = 53
write(1, "Name: Savkov2, Last Name: Igor2,"..., 53Name: Savkov2, Last Name: Igor2, Group: M8O-211-23
) = 53
write(1, "Name: Savkov3, Last Name: Igor3,"..., 53Name: Savkov3, Last Name: Igor3, Group: M8O-212-23
) = 53
write(1, "Name: Savkov4, Last Name: Igor4,"..., 53Name: Savkov4, Last Name: Igor4, Group: M8O-213-23
) = 53
write(1, "Name: Savkov1, Last Name: Igor1,"..., 53Name: Savkov1, Last Name: Igor1, Group: M8O-210-23
) = 53
write(1, "[LOG]Time to free block: 541 ns\n", 32[LOG]Time to free block: 541 ns
) = 32
write(1, "\n==============TEST2==========="..., 38
==============TEST2==============

) = 38
write(1, "[LOG]Time to allocate: 420 ns\n", 30[LOG]Time to allocate: 420 ns
) = 30
write(1, "array before mulp52:\n", 21array before mulp52:
) = 21
write(1, "long_double_array[0] = 1.000000\n", 32long_double_array[0] = 1.000000
) = 32
write(1, "long_double_array[1] = 2.000000\n", 32long_double_array[1] = 2.000000
) = 32
write(1, "long_double_array[2] = 3.000000\n", 32long_double_array[2] = 3.000000
) = 32

```
write(1, "long_double_array[3] = 4.000000\n", 32long_double_array[3] = 4.000000
) = 32
write(1, "long_double_array[4] = 5.000000\n", 32long_double_array[4] = 5.000000
) = 32
write(1, "long_double_array[5] = 6.000000\n", 32long_double_array[5] = 6.000000
) = 32
write(1, "long_double_array[6] = 7.000000\n", 32long_double_array[6] = 7.000000
) = 32
write(1, "long_double_array[7] = 8.000000\n", 32long_double_array[7] = 8.000000
) = 32
write(1, "long_double_array[8] = 9.000000\n", 32long_double_array[8] = 9.000000
) = 32
write(1, "long_double_array[9] = 10.000000"..., 33long_double_array[9] = 10.000000
) = 33
write(1, "long_double_array[10] = 11.00000"..., 34long_double_array[10] = 11.000000
) = 34
write(1, "long_double_array[11] = 12.00000"..., 34long_double_array[11] = 12.000000
) = 34
write(1, "long_double_array[12] = 13.00000"..., 34long_double_array[12] = 13.000000
) = 34
write(1, "long_double_array[13] = 14.00000"..., 34long_double_array[13] = 14.000000
) = 34
write(1, "long_double_array[14] = 15.00000"..., 34long_double_array[14] = 15.000000
) = 34
write(1, "long_double_array[15] = 16.00000"..., 34long_double_array[15] = 16.000000
) = 34
write(1, "long_double_array[16] = 17.00000"..., 34long_double_array[16] = 17.000000
) = 34
write(1, "long_double_array[17] = 18.00000"..., 34long_double_array[17] = 18.000000
) = 34
write(1, "long_double_array[18] = 19.00000"..., 34long_double_array[18] = 19.000000
) = 34
write(1, "long_double_array[19] = 20.00000"..., 34long_double_array[19] = 20.000000
) = 34
write(1, "array after mulp52:\n", 20array after mulp52:
)   = 20
write(1, "long_double_array[0] = 52.000000"..., 33long_double_array[0] = 52.000000
) = 33
write(1, "long_double_array[1] = 104.00000"..., 34long_double_array[1] = 104.000000
) = 34
write(1, "long_double_array[2] = 156.00000"..., 34long_double_array[2] = 156.000000
) = 34
write(1, "long_double_array[3] = 208.00000"..., 34long_double_array[3] = 208.000000
) = 34
write(1, "long_double_array[4] = 260.00000"..., 34long_double_array[4] = 260.000000
) = 34
write(1, "long_double_array[5] = 312.00000"..., 34long_double_array[5] = 312.000000
) = 34
write(1, "long_double_array[6] = 364.00000"..., 34long_double_array[6] = 364.000000
) = 34
```

```
write(1, "long_double_array[7] = 416.00000"..., 34long_double_array[7] = 416.000000
) = 34
write(1, "long_double_array[8] = 468.00000"..., 34long_double_array[8] = 468.000000
) = 34
write(1, "long_double_array[9] = 520.00000"..., 34long_double_array[9] = 520.000000
) = 34
write(1, "long_double_array[10] = 572.0000"..., 35long_double_array[10] = 572.000000
) = 35
write(1, "long_double_array[11] = 624.0000"..., 35long_double_array[11] = 624.000000
) = 35
write(1, "long_double_array[12] = 676.0000"..., 35long_double_array[12] = 676.000000
) = 35
write(1, "long_double_array[13] = 728.0000"..., 35long_double_array[13] = 728.000000
) = 35
write(1, "long_double_array[14] = 780.0000"..., 35long_double_array[14] = 780.000000
) = 35
write(1, "long_double_array[15] = 832.0000"..., 35long_double_array[15] = 832.000000
) = 35
write(1, "long_double_array[16] = 884.0000"..., 35long_double_array[16] = 884.000000
) = 35
write(1, "long_double_array[17] = 936.0000"..., 35long_double_array[17] = 936.000000
) = 35
write(1, "long_double_array[18] = 988.0000"..., 35long_double_array[18] = 988.000000
) = 35
write(1, "long_double_array[19] = 1040.000"..., 36long_double_array[19] = 1040.000000
) = 36
write(1, "[LOG]Time to free block: 1243 ns"..., 33[LOG]Time to free block: 1243 ns
) = 33
write(1, "\n==============TEST3==========="..., 38
==============TEST3==============

) = 38
write(1, "[LOG]Time to allocate: 962 ns\n", 30[LOG]Time to allocate: 962 ns
) = 30
write(1, "large_array[5000] = 52.000000\n", 30large_array[5000] = 52.000000
) = 30
write(1, "Large array (5001 long double) a"..., 64Large array (5001 long double) allocated and freed
successfully
) = 64
write(1, "[LOG]Time to free block: 220 ns\n", 32[LOG]Time to free block: 220 ns
) = 32
write(1, "\n==============TEST4==========="..., 38
==============TEST4==============

) = 38
write(1, "[LOG]Time to allocate: 281 ns\n", 30[LOG]Time to allocate: 281 ns
) = 30
write(1, "Memory allocated (3 * 1024 * 102"..., 41Memory allocated (3 * 1024 * 1024 bytes)
) = 41
write(1, "[LOG]Time to free block: 210 ns\n", 32[LOG]Time to free block: 210 ns
```

) = 32
write(1, "Memory freed (3 * 1024 * 1024 by"..., 37Memory freed (3 * 1024 * 1024 bytes)
) = 37
write(1, "Allocator destroyed\n", 20Allocator destroyed
) = 20
munmap(0x7fb3e9f8a000, 4194304) = 0
munmap(0x7fb3ea5b8000, 16424) = 0
exit_group(0) = ?
+++ exited with 0 +++

Вывод

В ходе работы были реализованы и протестированы два алгоритма аллокации памяти, которые сравнивались по фактору использования, скорости выделения и освобождения блоков, а также простоте использования. Оба аллокатора продемонстрировали эффективное использование пула памяти и интуитивность в применении, что подтвердило практическую ценность их реализации.