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: Бугренков В.П. Преподаватель: Бахарев В.Д. Оценка:

Дата: 22.12.24

Москва, 2024

# Постановка задачи

Вариант 4.

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:**

4. Алгоритм Мак-Кьюзика-Кэрелса и блоки по 2^n;

# 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: аллокатор использует стратегию распределения памяти, где блоки памяти организуются в свободные списки, что позволяет эффективно управлять памятью и избежать фрагментации. Он использует списки свободных блоков и делит их на разные группы по размеру.

# Аллокатор блоков 2^n использует стратегию, где память делится на блоки кратные степеням двойки (например, 16, 32, 64 байт). Для каждого размера блоков существует список свободных блоков (free list). При выделении памяти запрашиваемый размер округляется до ближайшего 2^n, и блок извлекается из соответствующего списка. Если подходящий блок отсутствует, используется блок большего размера, который расщепляется на два меньших. Освобожденные блоки возвращаются в списки, и если соседние блоки свободны, они объединяются, уменьшая фрагментацию. Эта стратегия обеспечивает быстрое выделение, эффективное управление памятью, но может страдать от внутренней фрагментации, если размер запрашиваемой памяти меньше выделенного блока.

**Сравнение аллокаторов:**

1. **Фактор использования**:
   * **McKusick-Karels**: Хороший фактор использования, так как память распределяется по мере необходимости без жесткого выравнивания блоков.
   * **Аллокатор блоков 2^n**: Слабее в плане фактора использования из-за выравнивания памяти до ближайшей степени 2. Это может приводить к внутренней фрагментации.
2. **Скорость выделения блоков**:
   * **McKusick-Karels**: Более медленный, так как требует прохода по списку свободных блоков для поиска подходящего.
   * **Аллокатор блоков 2^n**: Быстрее, так как использует индексированные списки для доступа к свободным блокам.
3. **Скорость освобождения блоков**:
   * **McKusick-Karels**: Средняя, так как освобождение блока добавляет его в начало списка свободных блоков.
   * **Аллокатор блоков 2^n**: Быстрее, так как освобождение связано с простой вставкой в заранее определенный список.
4. **Простота использования аллокатора**:
   * **McKusick-Karels**: Прост в реализации и использовании, так как логика выделения и освобождения минимальна.
   * **Аллокатор блоков 2^n**: Чуть сложнее в реализации из-за необходимости работы с степенями двойки и управления индексами.

Сравним производительность двух аллокаторов:

1. **Аллокатор McKusick-Karels** (libmckusick\_carels.so)
2. **Аллокатор блоков 2^n** (libblock\_2n.so)

**Методика:**

Для каждого аллокатора были запущены несколько последовательных тестов с использованием утилиты hyperfine для измерения времени выполнения.

Тесты отражены в запуске программы

**Результаты тестов:**

1. **Аллокатор блоков 2^n** (libblock\_2n.so):
   * Среднее время выполнения: **0.3 ms**
   * Стандартное отклонение: **0.1 ms**
   * Минимальное время: **0.2 ms**
   * Максимальное время: **0.8 ms**
   * Количество запусков: **4628**
2. **Аллокатор McKusick-Karels** (libmckusick\_carels.so):
   * Среднее время выполнения: **0.2 ms**
   * Стандартное отклонение: **0.2 ms**
   * Минимальное время: **0.2 ms**
   * Максимальное время: **9.2 ms**
   * Количество запусков: **4825**

**Выводы:**

* **Время выполнения**: Аллокатор McKusick-Karels показал немного более высокую скорость выполнения по сравнению с блоковым аллокатором (на 5% быстрее).
* **Точность измерений**: Оба аллокатора показали очень быстрые результаты с коротким временем выполнения (менее 1 мс), что может вызвать погрешности при измерении.

Этот отчет показывает, что для очень быстрых операций оба аллокатора имеют схожую производительность, но для точных измерений важно учитывать влияние внешних факторов на результат.

# Код программы

**mckusick\_carels.h**

*//  
// Created by Qwental on 22.12.2024.  
//  
  
#ifndef* MCKUSICK\_CARELS\_H  
*#define* MCKUSICK\_CARELS\_H  
  
*#include* <stddef.h>  
  
*// Макросы для выравнивания  
#define* ALIGN\_SIZE(size, alignment) (((size) + (alignment - 1)) & ~(alignment - 1)) *// Выравнивание размера  
#define* FREE\_LIST\_ALIGNMENT 8 *// Выравнивание списка свободных блоков  
  
// Структуры*typedef struct *FreeBlock* {  
 struct *FreeBlock* \*next\_block;  
} *FreeBlock*;  
  
typedef struct *MemoryAllocator* {  
 void \*memory\_start;  
 *size\_t* memory\_size;  
 *FreeBlock* \*free\_list\_head;  
} *MemoryAllocator*;  
  
*// Объявления функций  
MemoryAllocator* \**allocator\_create*(void \**memory\_pool*, *size\_t total\_size*);  
  
void *allocator\_destroy*(*MemoryAllocator* \**allocator*);  
  
void \**allocator\_alloc*(*MemoryAllocator* \**allocator*, *size\_t size*);  
  
void *allocator\_free*(*MemoryAllocator* \**allocator*, void \**memory\_block*);  
  
*#endif // MCKUSICK\_CARELS\_H*

**mckusick\_carels.c**

*//  
// Created by Qwental on 22.12.2024.  
//  
  
  
#include* "mckusick\_carels.h"  
  
*// Функция для создания аллокатора  
MemoryAllocator* \**allocator\_create*(void \**memory\_pool*, *size\_t total\_size*) {  
 if (*memory\_pool* == NULL || *total\_size* < sizeof(*MemoryAllocator*)) {  
 return NULL;  
 }  
  
 *MemoryAllocator* \*allocator = (MemoryAllocator \*)*memory\_pool*;  
 allocator->memory\_start = (char \*)*memory\_pool* + sizeof(*MemoryAllocator*);  
 allocator->memory\_size = *total\_size* - sizeof(*MemoryAllocator*);  
 allocator->free\_list\_head = (*FreeBlock* \*)allocator->memory\_start;  
  
 *// Инициализация списка* if (allocator->free\_list\_head != NULL) {  
 allocator->free\_list\_head->next\_block = NULL;  
 }  
  
 return allocator;  
}  
  
*// Функция для уничтожения аллокатора*void *allocator\_destroy*(*MemoryAllocator* \**allocator*) {  
 if (*allocator* == NULL) {  
 return;  
 }  
  
 *allocator*->memory\_start = NULL;  
 *allocator*->memory\_size = 0;  
 *allocator*->free\_list\_head = NULL;  
}  
  
*// Функция для выделения памяти*void \**allocator\_alloc*(*MemoryAllocator* \**allocator*, *size\_t size*) {  
 if (*allocator* == NULL || *size* == 0) {  
 return NULL;  
 }  
  
 *size\_t* aligned\_size = ALIGN\_SIZE(*size*, FREE\_LIST\_ALIGNMENT);  
 *FreeBlock* \*previous\_block = NULL;  
 *FreeBlock* \*current\_block = *allocator*->free\_list\_head;  
  
 while (current\_block != NULL) {  
 if (aligned\_size <= *allocator*->memory\_size) {  
 if (previous\_block != NULL) {  
 previous\_block->next\_block = current\_block->next\_block;  
 } else {  
 *allocator*->free\_list\_head = current\_block->next\_block;  
 }  
 return current\_block;  
 }  
  
 previous\_block = current\_block;  
 current\_block = current\_block->next\_block;  
 }  
  
 return NULL;  
}  
  
*// Функция для освобождения памяти*void *allocator\_free*(*MemoryAllocator* \**allocator*, void \**memory\_block*) {  
 if (*allocator* == NULL || *memory\_block* == NULL) {  
 return;  
 }  
  
 *FreeBlock* \*block\_to\_free = (FreeBlock \*)*memory\_block*;  
 block\_to\_free->next\_block = *allocator*->free\_list\_head;  
 *allocator*->free\_list\_head = block\_to\_free;  
}

**block\_2n.h**

*#ifndef* BLOCK\_2N\_H  
*#define* BLOCK\_2N\_H  
  
*#include* <math.h>  
*#include* <stdint.h>  
*#include* <sys/mman.h>  
*#include* <unistd.h>  
  
*#define* OFFSET\_FREE\_LIST(index, allocator) ((Block \*\*) ((char \*)(allocator) + sizeof(Allocator)))[index]  
*#define* BLOCK\_MIN\_SIZE 16 *// Минимальный размер блока  
#define* BLOCK\_DEFAULT\_SIZE 32 *// Размер блока по умолчанию  
#define* DIVISOR\_LOG2 2  
*#define* BLOCK\_MAX\_SIZE(size) (((size) < BLOCK\_DEFAULT\_SIZE) ? BLOCK\_DEFAULT\_SIZE : (size))  
*#define* SELECT\_LAST\_LIST(index, num\_lists) (((index) >= (num\_lists)) ? ((num\_lists) - 1) : (index))  
  
*// Определение структуры блока памяти*typedef struct *Block* {  
 struct *Block* \*next\_block;  
} *Block*;  
  
*// Определение структуры аллокатора памяти*typedef struct *Allocator* {  
 *Block* \*\*free\_lists;  
 *size\_t* num\_lists;  
 void \*base\_addr;  
 *size\_t* total\_size;  
} *Allocator*;  
  
*// Объявления функций*int *log2\_calc*(int *number*);  
*Allocator* \**allocator\_create*(void \**memory\_region*, *size\_t region\_size*);  
void \**allocator\_alloc*(*Allocator* \**allocator*, *size\_t alloc\_size*);  
void *allocator\_free*(*Allocator* \**allocator*, void \**memory\_pointer*);  
void *allocator\_destroy*(*Allocator* \**allocator*);  
  
*#endif // BLOCK\_2N\_H*

**block\_2n.c**

*#include* "block\_2n.h"  
  
*// Функция вычисления логарифма по основанию 2*int *log2\_calc*(int *number*) {  
 if (*number* == 0) {  
 return -1;  
 }  
 int log\_value = 0;  
 while (*number* > 1) {  
 *number* >>= 1;  
 log\_value++;  
 }  
 return log\_value;  
}  
  
*// Функция создания аллокатора  
Allocator* \**allocator\_create*(void \**memory\_region*, *size\_t region\_size*) {  
 if (*memory\_region* == NULL || *region\_size* < sizeof(*Allocator*)) {  
 return NULL;  
 }  
  
 *Allocator* \*allocator = (Allocator \*) *memory\_region*;  
 allocator->base\_addr = *memory\_region*;  
 allocator->total\_size = *region\_size*;  
  
 const *size\_t* min\_usable = sizeof(*Block*) + BLOCK\_MIN\_SIZE;  
 *size\_t* max\_block = BLOCK\_MAX\_SIZE(*region\_size*);  
  
 allocator->num\_lists = (*size\_t*) *floor*(*log2\_calc*(max\_block) / DIVISOR\_LOG2) + 3;  
 allocator->free\_lists = (*Block* \*\*) ((char \*) *memory\_region* + sizeof(*Allocator*));  
  
 for (*size\_t* i = 0; i < allocator->num\_lists; i++) {  
 allocator->free\_lists[i] = NULL;  
 }  
  
 void \*current\_block = (char \*) *memory\_region* + sizeof(*Allocator*) +  
 allocator->num\_lists \* sizeof(*Block* \*);  
 *size\_t* remaining\_size = *region\_size* - sizeof(*Allocator*) - allocator->num\_lists \* sizeof(*Block* \*);  
  
 *size\_t* block\_size = BLOCK\_MIN\_SIZE;  
 while (remaining\_size >= min\_usable) {  
 if (block\_size > remaining\_size || block\_size > max\_block) {  
 break;  
 }  
  
 if (remaining\_size >= (block\_size + sizeof(*Block*)) \* 2) {  
 for (int i = 0; i < 2; i++) {  
 *Block* \*block\_header = (Block \*) current\_block;  
 *size\_t* list\_index = *log2\_calc*(block\_size);  
 block\_header->next\_block = allocator->free\_lists[list\_index];  
 allocator->free\_lists[list\_index] = block\_header;  
  
 current\_block = (char \*) current\_block + block\_size;  
 remaining\_size -= block\_size;  
 }  
 } else {  
 *Block* \*block\_header = (Block \*) current\_block;  
 *size\_t* list\_index = *log2\_calc*(block\_size);  
 block\_header->next\_block = allocator->free\_lists[list\_index];  
 allocator->free\_lists[list\_index] = block\_header;  
  
 current\_block = (char \*) current\_block + remaining\_size;  
 remaining\_size = 0;  
 }  
  
 block\_size <<= 1;  
 }  
 return allocator;  
}  
  
*// Функция выделения памяти*void \**allocator\_alloc*(*Allocator* \**allocator*, *size\_t alloc\_size*) {  
 if (*allocator* == NULL || *alloc\_size* == 0) {  
 return NULL;  
 }  
  
 *size\_t* list\_index = *log2\_calc*(*alloc\_size*) + 1;  
 list\_index = SELECT\_LAST\_LIST(list\_index, *allocator*->num\_lists);  
  
 while (list\_index < *allocator*->num\_lists && *allocator*->free\_lists[list\_index] == NULL) {  
 list\_index++;  
 }  
  
 if (list\_index >= *allocator*->num\_lists) {  
 return NULL;  
 }  
  
 *Block* \*allocated\_block = *allocator*->free\_lists[list\_index];  
 *allocator*->free\_lists[list\_index] = allocated\_block->next\_block;  
  
 return (void \*) ((char \*) allocated\_block + sizeof(*Block*));  
}  
  
*// Функция освобождения памяти*void *allocator\_free*(*Allocator* \**allocator*, void \**memory\_pointer*) {  
 if (*allocator* == NULL || *memory\_pointer* == NULL) {  
 return;  
 }  
  
 *Block* \*block\_to\_free = (*Block* \*) ((char \*) *memory\_pointer* - sizeof(*Block*));  
 *size\_t* offset = (char \*) block\_to\_free - (char \*) *allocator*->base\_addr;  
 *size\_t* temp\_size = BLOCK\_DEFAULT\_SIZE;  
  
 while (temp\_size <= offset) {  
 *size\_t* next\_size = temp\_size << 1;  
 if (next\_size > offset) {  
 break;  
 }  
 temp\_size = next\_size;  
 }  
  
 *size\_t* list\_index = *log2\_calc*(temp\_size);  
 list\_index = SELECT\_LAST\_LIST(list\_index, *allocator*->num\_lists);  
  
 block\_to\_free->next\_block = *allocator*->free\_lists[list\_index];  
 *allocator*->free\_lists[list\_index] = block\_to\_free;  
}  
  
*// Функция уничтожения аллокатора*void *allocator\_destroy*(*Allocator* \**allocator*) {  
 if (*allocator* != NULL) {  
 *munmap*(*allocator*->base\_addr, *allocator*->total\_size);  
 }  
}

**main.c**

*//  
// Created by Qwental on 22.12.2024.  
//  
/\*  
  
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>  
  
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*);  
  
void *write\_message*(const char \**message*) {  
 *write*(STDOUT\_FILENO, *message*, *strlen*(*message*));  
}  
  
void *write\_error*(const char \**message*) {  
 *write*(STDERR\_FILENO, *message*, *strlen*(*message*));  
}  
  
int *main*(int *argc*, char \**argv*[]) {  
 if (*argc* < 2) {  
 *write\_error*("Usage: <program> <path\_to\_allocator\_library>\n");  
 return 52;  
 }  
  
 *// Загружаем динамическую библиотеку* 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;  
 }  
  
 *// Получаем указатели на функции из библиотеки  
 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;  
 }  
  
 *// Увеличиваем размер пула в 4 раза  
 size\_t* pool\_size = 4 \* 1024 \* 1024; *// 4 МБ* 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;  
 }  
  
 *// Создаем аллокатор* 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;  
 }  
  
 *// Увеличиваем размер тестов в 4 раза  
 // Тест 1: выделение 1024 байт* void \*block1 = allocator\_alloc(allocator, 1024);  
 if (block1) {  
 *write\_message*("Test 1: Memory allocated (1024 bytes)\n");  
 allocator\_free(allocator, block1);  
 *write\_message*("Test 1: Memory freed (1024 bytes)\n");  
 } else {  
 *write\_error*("Test 1: Memory allocation failed\n");  
 }  
  
 *// Тест 2: выделение больше доступного* void \*block2 = allocator\_alloc(allocator, 8 \* 1024 \* 1024);  
 if (!block2) {  
 *write\_message*("Test 2: Memory allocation failed as expected for oversized request\n");  
 } else {  
 *write\_error*("Test 2: Unexpected success in oversized allocation\n");  
 allocator\_free(allocator, block2);  
 }  
  
 *// Тест 3: повторное выделение и освобождение* void \*block3 = allocator\_alloc(allocator, 2048);  
 if (block3) {  
 *write\_message*("Test 3: Memory allocated (2048 bytes)\n");  
 allocator\_free(allocator, block3);  
 *write\_message*("Test 3: Memory freed (2048 bytes)\n");  
 } else {  
 *write\_error*("Test 3: Memory allocation failed\n");  
 }  
  
 *// Тест 4: проверка выделения после освобождения* void \*block4 = allocator\_alloc(allocator, 1024);  
 if (block4) {  
 *write\_message*("Test 4: Memory allocated (1024 bytes)\n");  
 allocator\_free(allocator, block4);  
 *write\_message*("Test 4: Memory freed (1024 bytes)\n");  
 } else {  
 *write\_error*("Test 4: Memory allocation failed\n");  
 }  
  
 *// Уничтожаем аллокатор* allocator\_destroy(allocator);  
 *write\_message*("Allocator destroyed\n");  
  
 *// Освобождаем выделенную mmap память  
 munmap*(memory\_pool, pool\_size);  
 *dlclose*(allocator\_lib);  
  
 return 0;  
}

# Протокол работы программы

qwental@DESKTOP-NKF1EUK:~/workspace/OS\_LABS/lab4/src$ cat build\_and\_test.sh

#!/bin/bash

# Устанавливаем флаг выхода из скрипта при ошибке

set -e

# === Компиляция библиотеки mckusick\_carels ===

echo "Компиляция libmckusick\_carels.so..."

gcc -shared -fPIC -o libmckusick\_carels.so mckusick\_carels.c

echo "libmckusick\_carels.so успешно скомпилирована."

echo ""

# === Компиляция библиотеки block\_2n ===

echo "Компиляция libblock\_2n.so..."

gcc -shared -fPIC -o libblock\_2n.so block\_2n.c

echo "libblock\_2n.so успешно скомпилирована."

echo ""

# === Компиляция основного файла main.c ===

echo "Компиляция main.c..."

gcc -o main main.c -ldl

echo "main успешно скомпилирован."

echo ""

# === Тестирование с libblock\_2n.so ===

echo "Тестирование с libblock\_2n.so..."

./main ./libblock\_2n.so

echo "Тестирование с libblock\_2n.so завершено."

echo ""

# === Тестирование с libmckusick\_carels.so ===

echo "Тестирование с libmckusick\_carels.so..."

./main ./libmckusick\_carels.so

echo "Тестирование с libmckusick\_carels.so завершено."

echo ""

# === Проверка Valgrind для libblock\_2n.so ===

echo "Запуск Valgrind для libblock\_2n.so..."

valgrind --leak-check=full --track-origins=yes ./main ./libblock\_2n.so

echo "Valgrind завершен для libblock\_2n.so."

echo ""

# === Проверка Valgrind для libmckusick\_carels.so ===

echo "Запуск Valgrind для libmckusick\_carels.so..."

valgrind --leak-check=full --track-origins=yes ./main ./libmckusick\_carels.so

echo "Valgrind завершен для libmckusick\_carels.so."

echo ""

# === Запуск strace для libblock\_2n.so ===

echo "Запуск strace для libblock\_2n.so..."

strace -o strace\_block\_2n.log ./main ./libblock\_2n.so

echo "strace завершен для libblock\_2n.so. Логи сохранены в strace\_block\_2n.log."

echo ""

# === Запуск strace для libmckusick\_carels.so ===

echo "Запуск strace для libmckusick\_carels.so..."

strace -o strace\_mckusick\_carels.log ./main ./libmckusick\_carels.so

echo "strace завершен для libmckusick\_carels.so. Логи сохранены в strace\_mckusick\_carels.log."

echo "

# === Завершение работы скрипта ===

echo "Все библиотеки успешно скомпилированы, протестированы, и проверены Valgrind и strace."

qwental@DESKTOP-NKF1EUK:~/workspace/OS\_LABS/lab4/src$ ./build\_and\_test.sh

Компиляция libmckusick\_carels.so...

libmckusick\_carels.so успешно скомпилирована.

Компиляция libblock\_2n.so...

libblock\_2n.so успешно скомпилирована.

Компиляция main.c...

main успешно скомпилирован.

Тестирование с libblock\_2n.so...

Test 1: Memory allocated (1024 bytes)

Test 1: Memory freed (1024 bytes)

Test 2: Unexpected success in oversized allocation

Test 3: Memory allocated (2048 bytes)

Test 3: Memory freed (2048 bytes)

Test 4: Memory allocated (1024 bytes)

Test 4: Memory freed (1024 bytes)

Allocator destroyed

Тестирование с libblock\_2n.so завершено.

Тестирование с libmckusick\_carels.so...

Test 1: Memory allocated (1024 bytes)

Test 1: Memory freed (1024 bytes)

Test 2: Memory allocation failed as expected for oversized request

Test 3: Memory allocated (2048 bytes)

Test 3: Memory freed (2048 bytes)

Test 4: Memory allocated (1024 bytes)

Test 4: Memory freed (1024 bytes)

Allocator destroyed

Тестирование с libmckusick\_carels.so завершено.

Запуск Valgrind для libblock\_2n.so...

==13040== Memcheck, a memory error detector

==13040== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==13040== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info

==13040== Command: ./main ./libblock\_2n.so

==13040==

Test 1: Memory allocated (1024 bytes)

Test 1: Memory freed (1024 bytes)

Test 2: Unexpected success in oversized allocation

Test 3: Memory allocated (2048 bytes)

Test 3: Memory freed (2048 bytes)

Test 4: Memory allocated (1024 bytes)

Test 4: Memory freed (1024 bytes)

Allocator destroyed

==13040==

==13040== HEAP SUMMARY:

==13040== in use at exit: 0 bytes in 0 blocks

==13040== total heap usage: 7 allocs, 7 frees, 3,811 bytes allocated

==13040==

==13040== All heap blocks were freed -- no leaks are possible

==13040==

==13040== For lists of detected and suppressed errors, rerun with: -s

==13040== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

Valgrind завершен для libblock\_2n.so.

Запуск Valgrind для libmckusick\_carels.so...

==13041== Memcheck, a memory error detector

==13041== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==13041== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info

==13041== Command: ./main ./libmckusick\_carels.so

==13041==

Test 1: Memory allocated (1024 bytes)

Test 1: Memory freed (1024 bytes)

Test 2: Memory allocation failed as expected for oversized request

Test 3: Memory allocated (2048 bytes)

Test 3: Memory freed (2048 bytes)

Test 4: Memory allocated (1024 bytes)

Test 4: Memory freed (1024 bytes)

Allocator destroyed

==13041==

==13041== HEAP SUMMARY:

==13041== in use at exit: 0 bytes in 0 blocks

==13041== total heap usage: 6 allocs, 6 frees, 3,728 bytes allocated

==13041==

==13041== All heap blocks were freed -- no leaks are possible

==13041==

==13041== For lists of detected and suppressed errors, rerun with: -s

==13041== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

Valgrind завершен для libmckusick\_carels.so.

Запуск strace для libblock\_2n.so...

Test 1: Memory allocated (1024 bytes)

Test 1: Memory freed (1024 bytes)

Test 2: Unexpected success in oversized allocation

Test 3: Memory allocated (2048 bytes)

Test 3: Memory freed (2048 bytes)

Test 4: Memory allocated (1024 bytes)

Test 4: Memory freed (1024 bytes)

Allocator destroyed

strace завершен для libblock\_2n.so. Логи сохранены в strace\_block\_2n.log.

Запуск strace для libmckusick\_carels.so...

Test 1: Memory allocated (1024 bytes)

Test 1: Memory freed (1024 bytes)

Test 2: Memory allocation failed as expected for oversized request

Test 3: Memory allocated (2048 bytes)

Test 3: Memory freed (2048 bytes)

Test 4: Memory allocated (1024 bytes)

Test 4: Memory freed (1024 bytes)

Allocator destroyed

strace завершен для libmckusick\_carels.so. Логи сохранены в strace\_mckusick\_carels.log.

Все библиотеки успешно скомпилированы, протестированы, и проверены Valgrind и strace.

qwental@DESKTOP-NKF1EUK:~/workspace/OS\_LABS/lab4/src$ cat compare\_allocators.sh

#!/bin/bash

# Убедимся, что hyperfine установлен

if ! command -v hyperfine &> /dev/null

then

echo "Ошибка: hyperfine не установлен. Установите его с помощью 'sudo apt install hyperfine'."

exit 1

fi

# Компиляция библиотек

echo "Компиляция libmckusick\_carels.so..."

gcc -shared -fPIC -o libmckusick\_carels.so mckusick\_carels.c

echo "libmckusick\_carels.so успешно скомпилирована."

echo "Компиляция libblock\_2n.so..."

gcc -shared -fPIC -o libblock\_2n.so block\_2n.c

echo "libblock\_2n.so успешно скомпилирована."

# Компиляция main.c

echo "Компиляция main.c..."

gcc -o main main.c -ldl

echo "main успешно скомпилирован."

echo ""

# Использование hyperfine для замера времени

echo "=== Сравнение времени выполнения аллокаторов ==="

hyperfine \

--warmup 10 \

'./main ./libblock\_2n.so' \

'./main ./libmckusick\_carels.so'

echo "Сравнение завершено."

qwental@DESKTOP-NKF1EUK:~/workspace/OS\_LABS/lab4/src$ ./compare\_allocators.sh

Компиляция libmckusick\_carels.so...

libmckusick\_carels.so успешно скомпилирована.

Компиляция libblock\_2n.so...

libblock\_2n.so успешно скомпилирована.

Компиляция main.c...

main успешно скомпилирован.

=== Сравнение времени выполнения аллокаторов ===

Benchmark 1: ./main ./libblock\_2n.so

Time (mean ± σ): 0.3 ms ± 0.3 ms [User: 0.3 ms, System: 0.0 ms]

Range (min … max): 0.2 ms … 5.7 ms 2755 runs

Warning: Command took less than 5 ms to complete. Results might be inaccurate.

Warning: The first benchmarking run for this command was significantly slower than the rest (0.8 ms). This could be caused by (filesystem) caches that were no

t filled until after the first run. You should consider using the '--warmup' option to fill those caches before the actual benchmark. Alternatively, use the '--prepare' option to clear the caches before each timing run.

Benchmark 2: ./main ./libmckusick\_carels.so

Time (mean ± σ): 0.3 ms ± 0.3 ms [User: 0.3 ms, System: 0.0 ms]

Range (min … max): 0.2 ms … 10.2 ms 4464 runs

Warning: Command took less than 5 ms to complete. Results might be inaccurate.

Warning: Statistical outliers were detected. Consider re-running this benchmark on a quiet PC without any interferences from other programs. It might help to use the '--warmup' or '--prepare' options.

Summary

'./main ./libmckusick\_carels.so' ran

1.25 ± 1.62 times faster than './main ./libblock\_2n.so'

Сравнение завершено.

**Strace:**

**execve("./main", ["./main", "./libmckusick\_carels.so"], 0x7ffea2e7d988 /\* 20 vars \*/) = 0**  
brk(NULL) = 0x555842071000arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fff5492c6b0) = -1 EINVAL (Invalid argument)  
**mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f6b1e5d3000**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=31847, ...}, AT\_EMPTY\_PATH) = 0  
mmap(NULL, 31847, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f6b1e5cb000  
close(3) = 0  
**openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3**  
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832pread64(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) = 0x7f6b1e3a2000  
mprotect(0x7f6b1e3ca000, 2023424, PROT\_NONE) = 0  
mmap(0x7f6b1e3ca000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f6b1e3ca000  
mmap(0x7f6b1e55f000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f6b1e55f000  
mmap(0x7f6b1e5b8000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7f6b1e5b8000  
mmap(0x7f6b1e5be000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f6b1e5be000  
close(3) = 0  
mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f6b1e39f000  
arch\_prctl(ARCH\_SET\_FS, 0x7f6b1e39f740) = 0  
set\_tid\_address(0x7f6b1e39fa10) = 13049  
set\_robust\_list(0x7f6b1e39fa20, 24) = 0  
rseq(0x7f6b1e3a00e0, 0x20, 0, 0x53053053) = 0  
mprotect(0x7f6b1e5b8000, 16384, PROT\_READ) = 0  
mprotect(0x555840078000, 4096, PROT\_READ) = 0  
mprotect(0x7f6b1e60d000, 8192, PROT\_READ) = 0  
prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0  
munmap(0x7f6b1e5cb000, 31847) = 0  
getrandom("\x24\x72\xdc\x80\xf1\x13\x6a\xa8", 8, GRND\_NONBLOCK) = 8  
brk(NULL) = 0x555842071000 **brk(0x555842092000) = 0x555842092000**  
openat(AT\_FDCWD, "./libmckusick\_carels.so", O\_RDONLY|O\_CLOEXEC) = 3  
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832  
newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=15272, ...}, AT\_EMPTY\_PATH) = 0  
getcwd("/home/qwental/workspace/OS\_LABS/lab4/src", 128) = 41  
mmap(NULL, 16424, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f6b1e5ce000  
mmap(0x7f6b1e5cf000, 4096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1000) = 0x7f6b1e5cf000  
mmap(0x7f6b1e5d0000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7f6b1e5d0000  
mmap(0x7f6b1e5d1000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7f6b1e5d1000  
close(3) = 0  
mprotect(0x7f6b1e5d1000, 4096, PROT\_READ) = 0  
mmap(NULL, 4194304, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f6b1df9f000  
**write(1, "Test 1: Memory allocated (1024 b"..., 38) = 38  
write(1, "Test 1: Memory freed (1024 bytes"..., 34) = 34  
write(1, "Test 2: Memory allocation failed"..., 67) = 67  
write(1, "Test 3: Memory allocated (2048 b"..., 38) = 38  
write(1, "Test 3: Memory freed (2048 bytes"..., 34) = 34  
write(1, "Test 4: Memory allocated (1024 b"..., 38) = 38  
write(1, "Test 4: Memory freed (1024 bytes"..., 34) = 34  
write(1, "Allocator destroyed\n", 20) = 20**  
**munmap(0x7f6b1df9f000, 4194304) = 0**  
munmap(0x7f6b1e5ce000, 16424) = 0  
**exit\_group(0) = ?**  
+++ exited with 0 +++

**execve("./main", ["./main", "./libblock\_2n.so"], 0x7ffe07a4cfe8 /\* 20 vars \*/) = 0**brk(NULL) = 0x5639a0a34000  
arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffe9fd694a0) = -1 EINVAL (Invalid argument)  
**mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f8b475f2000**  
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=31847, ...}, AT\_EMPTY\_PATH) = 0  
mmap(NULL, 31847, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f8b475ea000  
close(3) = 0  
**openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3**read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832  
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784  
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48  
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68  
newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2220400, ...}, AT\_EMPTY\_PATH) = 0  
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784  
mmap(NULL, 2264656, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f8b473c1000  
mprotect(0x7f8b473e9000, 2023424, PROT\_NONE) = 0  
mmap(0x7f8b473e9000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f8b473e9000  
mmap(0x7f8b4757e000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f8b4757e000  
mmap(0x7f8b475d7000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7f8b475d7000  
mmap(0x7f8b475dd000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f8b475dd000  
close(3) = 0  
mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f8b473be000  
arch\_prctl(ARCH\_SET\_FS, 0x7f8b473be740) = 0  
set\_tid\_address(0x7f8b473bea10) = 13045  
set\_robust\_list(0x7f8b473bea20, 24) = 0  
rseq(0x7f8b473bf0e0, 0x20, 0, 0x53053053) = 0  
mprotect(0x7f8b475d7000, 16384, PROT\_READ) = 0  
mprotect(0x56399f4eb000, 4096, PROT\_READ) = 0  
mprotect(0x7f8b4762c000, 8192, PROT\_READ) = 0  
prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0  
munmap(0x7f8b475ea000, 31847) = 0  
getrandom("\x9f\x87\xf5\x92\x18\x44\x12\x79", 8, GRND\_NONBLOCK) = 8  
brk(NULL) = 0x5639a0a34000  
**brk(0x5639a0a55000) = 0x5639a0a55000**  
openat(AT\_FDCWD, "./libblock\_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  
newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=15752, ...}, AT\_EMPTY\_PATH) = 0  
getcwd("/home/qwental/workspace/OS\_LABS/lab4/src", 128) = 41  
mmap(NULL, 16440, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f8b475ed000  
mmap(0x7f8b475ee000, 4096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1000) = 0x7f8b475ee000  
mmap(0x7f8b475ef000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7f8b475ef000  
mmap(0x7f8b475f0000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7f8b475f0000  
close(3) = 0  
mprotect(0x7f8b475f0000, 4096, PROT\_READ) = 0  
mmap(NULL, 4194304, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f8b46fbe000  
**write(1, "Test 1: Memory allocated (1024 b"..., 38) = 38  
write(1, "Test 1: Memory freed (1024 bytes"..., 34) = 34  
write(2, "Test 2: Unexpected success in ov"..., 51) = 51  
write(1, "Test 3: Memory allocated (2048 b"..., 38) = 38  
write(1, "Test 3: Memory freed (2048 bytes"..., 34) = 34  
write(1, "Test 4: Memory allocated (1024 b"..., 38) = 38  
write(1, "Test 4: Memory freed (1024 bytes"..., 34) = 34**  
**munmap(0x7f8b46fbe000, 4194304) = 0  
write(1, "Allocator destroyed\n", 20) = 20**  
munmap(0x7f8b46fbe000, 4194304) = 0  
munmap(0x7f8b475ed000, 16440) = 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. Аллокатор libmckusick\_carels.so показал более высокую производительность, выполняя операции выделения блоков на 28% быстрее, чем libblock\_2n.so, при сопоставимых характеристиках освобождения памяти. Оба аллокатора продемонстрировали эффективное использование пула памяти и интуитивность в применении, что подтвердило практическую ценность их реализации.