Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа по курсу**

**«Операционные системы»**

Группа: М8О-209Б-22

Студент: Концебалов О.С.

Преподаватель: Пономарев Н.В.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 25.12.2023

Москва, 2023.

# Содержание

1. Постановка задачи.
2. Общие сведения о программе.
3. Общий метод и алгоритм решения.
4. Код программы.
5. Демонстрация работы программы.
6. Вывод.

## Постановка задачи

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Провести анализ, сделать вывод на основании данных, полученных при работе программного прототипа.

Вариант курсового проекта: Аллокаторы памяти

Исследование 2 аллокаторов памяти: необходимо реализовать два алгоритма аллокации памяти и сравнить их по следующим характеристикам:  
• Фактор использования  
• Скорость выделения блоков  
• Скорость освобождения блоков  
• Простота использования аллокатора  
 Каждый аллокатор памяти должен иметь функции аналогичные стандартным функциям free и malloc (realloc, опционально). Перед работой каждый аллокатор инициализируется свободными страницами памяти, выделенными стандартными средствами ядра. 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.  
 В отчете необходимо отобразить следующее:  
• Подробное описание каждого из исследуемых алгоритмов  
• Процесс тестирования  
• Обоснование подхода тестирования  
• Результаты тестирования  
• Заключение по проведенной работе

Сравнение алгоритмов аллокаторов памяти (детальное описание задания описано выше). Каждый аллокатор должен обладать следующим интерфейсом (могут быть отличия в зависимости от особенностей алгоритма):  
• Allocator\* createMemoryAllocator(void \*realMemory, size\_t memory\_size) (создание  
аллокатора памяти размера memory\_size)  
• void\* alloc(Allocator \* allocator, size\_t block\_size) (выделение памяти при помощи  
аллокатора размера block\_size)  
• void\* free(Allocator \* allocator, void \* block) (возвращает выделенную память аллокатору)

Задание варианта: Необходимо сравнить два алгоритма аллокации: списки свободных блоков (первое подходящее) и алгоритм Мак-Кьюзи-Кэрелса.

## Общие сведения о программе

Программа состоит из двух папок include/allocator и src. В папке include/allocator находятся три заголовочных файла: IAllocator.h – интерфейсный класс Allocator, который является чисто виртуальным, и от которого будут отнаследованы два других класса Аллокаторов, каждый из которых использует опеределенный алгоритм аллокации; ListAllocator.h – класс аллокатора, который использует алгоритм списка свободных блоков (первое подходящее); MacKuseyCarelsAllocator.h – класс аллокатора, который использует алгоритм Мак-Кьюзи-Кэрелса. В папке src находятся две папки: allocator, в которой находятся два файла ListAllocator.cpp и MacKuseyCarelsAllocator.cpp с реализацией соответствующих классов, и папки benchmark, в которой находится файл benchmark.cpp для сравнения двух алгоритмов аллокации. Также в главной директории проекта находится файл run.cpp для запуска кода, Makefile и CMakeLists.txt для удобной сборки и запуска проекта.

Для реализации класса ListAllocator.h использовался алгоритм «Списки свободных блоков (первое подходящее)».

Алгоритм аллокации "Списки свободных блоков (первое подходящее)" используется для эффективной работы с динамической памятью. Основная идея этого алгоритма заключается в том, чтобы иметь список свободных блоков памяти и выбирать первый блок, который удовлетворяет требованиям размера запрашиваемой памяти.

1. Инициализация памяти: изначально, имеется единый большой блок памяти, который целиком доступен для распределения. Этот блок инициализируется как "свободный".

2. Структура данных: для хранения списка свободных блоков, часто используется связный список. Каждый элемент списка содержит информацию о размере блока и указатель на следующий свободный блок.

3. Аллокация памяти: при запросе аллокации памяти определенного размера, алгоритм просматривает список свободных блоков, начиная с самого начала списка. Если находится блок, размер которого достаточно большой, чтобы удовлетворить запрос, то этот блок используется для аллокации. Для этого блоку просто присваивается статус "занят", и происходит выдача указателя на его начало. Если блок недостаточно большой, алгоритм продолжает поиск следующего блока. Если список свободных блоков заканчивается и подходящий блок не найден, может быть выполнено управление остатком памяти.

4. Освобождение памяти: при освобождении блока памяти, он помечается как "свободный". Затем алгоритм проверяет соседние свободные блоки. Если они существуют, то происходит их объединение в один большой блок. Это позволяет эффективно использовать фрагментированную память, устраняя разрывы между свободными блоками.

Преимущества алгоритма "Списки свободных блоков (первое подходящее)":

- Простота реализации и понимания.

- Эффективное использование памяти за счет объединения свободных блоков.

- Быстрый доступ к первому подходящему блоку.

Недостатки алгоритма "Списки свободных блоков (первое подходящее)":

- Возможна фрагментация памяти при неправильном распределении блоков.

- Поиск подходящего блока может занимать больше времени, если он находится в конце списка свободных блоков.

- Требуется дополнительное пространство для хранения указателей на следующий блок в списке свободных блоков.

В целом, алгоритм аллокации "Списки свободных блоков (первое подходящее)" является простым и эффективным способом управления памятью во время выполнения программы. Он широко применяется в различных системах и является одним из основных алгоритмов аллокации памяти.

Для реализации класса MacKuseyCarelsAllocator.h использовался алгоритм аллокации «Мак-Кьюзи-Кэрелса».

Алгоритм аллокации Мак-Кьюзи-Кэрелса (MCC) является одним из способов управления динамической памятью и используется в системах, где имеется ограниченное количество свободного пространства и требуется эффективное использование памяти.

1. Инициализация памяти: при запуске алгоритма имеется единый большой блок памяти, который целиком доступен для распределения. Этот блок инициализируется как "свободный".

2. Разбиение блоков: изначально все свободное пространство представлено в виде одного блока. При аллокации памяти блок может быть разделен на два более маленьких блока: один используется для запрошенного объема памяти, а второй остается свободным.

3. Структура данных: для хранения информации о блоках памяти используются двусвязные списки. Каждый блок содержит информацию о размере и состоянии (занят или свободен), а также указатели на предыдущий и следующий блоки.

4. Аллокация памяти: при запросе аллокации памяти определенного размера, алгоритм просматривает связанный список свободных блоков, начиная с самого начала списка. Если находится свободный блок, размер которого достаточно большой, чтобы удовлетворить запрос, то этот блок используется для аллокации. Для этого блок помечается как "занятый", а указатель на начало блока возвращается как результат аллокации. Если блок недостаточно большой, алгоритм продолжает поиск в списке свободных блоков. Если блоки заканчиваются и подходящий блок не найден, происходит обработка ошибки.

5. Освобождение памяти: при освобождении блока памяти, он помечается как "свободный". Затем алгоритм проверяет блоки-соседи. Если соседние блоки также являются свободными, происходит их объединение в один большой блок. Это позволяет эффективно использовать фрагментированную память, устраняя разрывы между свободными блоками.

Преимущества алгоритма Мак-Кьюзи-Кэрелса:

- Устранение фрагментации памяти путем объединения свободных блоков.

- Поддержка размещения памяти переменного размера.

- Более эффективное использование памяти по сравнению с другими алгоритмами.

Недостатки алгоритма Мак-Кьюзи-Кэрелса:

- Дополнительное пространство требуется для хранения информации о каждом блоке.

- Поиск подходящего блока может занимать больше времени, особенно если список свободных блоков длинный.

- Нет возможности выделения памяти, которая является частью свободного блока без разделения его на несколько блоков.

В целом, алгоритм аллокации Мак-Кьюзи-Кэрелса представляет собой эффективный способ управления динамической памятью, особенно в условиях ограниченного пространства. Он широко применяется в различных программных системах, где требуется эффективное использование памяти и управление динамическим выделением памяти.

## Общий метод и алгоритм решения

Алгоритмы аллокации, а также суть их реализации описаны выше. Для сравнения двух алгоритмов аллокации мной был написан специальный бенчмарк. Сравнивал алгоритмы я по самому важному критерию – затраченному времени на аллокацию. Для этой цели использовалась библиотека chrono, с ее широким функционалом для замеров времени. Я делал очень большое количество запросов на выделение памяти и на освобождение и смотрел на поведение программы и затраченное время.

Код был написан в парадигме ООП с использованием наследования и полиморфизма.

Для разработки классов Аллокаторов использовал cppreference и статью allocatortraits. Очень помогли

## Код программы

./include/allocator/IAllocator.h

#pragma once

#include <exception>

#include <iostream>

#include <math.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <stdbool.h>

#include <sys/mman.h>

namespace allocator {

class Allocator {

  public:

    using void\_pointer = void\*;

    using size\_type = std::size\_t;

    using difference\_type = std::ptrdiff\_t;

    using propagate\_on\_container\_move\_assignment = std::true\_type;

    using is\_always\_equal = std::true\_type;

  protected:

    Allocator() = default;

  public:

    virtual ~Allocator() = default;

    virtual void\_pointer alloc(const size\_type) = 0;

    virtual void free(void\_pointer) = 0;

};

}; // namespace allocator

./include/allocator/ListAllocator.h

#pragma once

#include "IAllocator.h"

namespace allocator {

struct BlockHeader {

    size\_t \_size;

    BlockHeader\* \_next;

};

class ListAllocator final : public Allocator {

  private:

    BlockHeader\* \_free\_blocks\_list;

  public:

    ListAllocator() = delete;

    ListAllocator(void\_pointer, size\_type);

    virtual ~ListAllocator();

    virtual void\_pointer alloc(size\_type) override;

    virtual void free(void\_pointer) override;

};

}; // namespace allocator

./include/allocator/ MacKuseyCarelsAllocator.h

#pragma once

#include "IAllocator.h"

namespace allocator {

struct Page {

    Page\* \_next;

    bool \_is\_large;

    size\_t \_block\_size;

};

class MacKuseyCarelsAllocator final : public Allocator {

  private:

    void\* \_memory;

    Page\* \_free\_pages\_list;

    size\_t \_memory\_size;

    size\_t \_page\_size;

  public:

    MacKuseyCarelsAllocator() = delete;

    MacKuseyCarelsAllocator(void\_pointer, size\_type);

    virtual ~MacKuseyCarelsAllocator();

    virtual void\_pointer alloc(size\_type) override;

    virtual void free(void\_pointer) override;

};

}; // namespace allocator

./src/allocator/ListAllocator.cpp

#include "../../include/allocator/ListAllocator.h"

using namespace allocator;

ListAllocator::ListAllocator(void\_pointer real\_memory, size\_type memory\_size)

{

    \_free\_blocks\_list = reinterpret\_cast<BlockHeader\*>(real\_memory + sizeof(ListAllocator));

    \_free\_blocks\_list->\_size = memory\_size - sizeof(ListAllocator) - sizeof(BlockHeader);

    \_free\_blocks\_list->\_next = nullptr;

}

ListAllocator::~ListAllocator()

{

    BlockHeader\* cur\_block = this->\_free\_blocks\_list;

    while (cur\_block) {

        BlockHeader\* to\_delete = cur\_block;

        cur\_block = cur\_block->\_next;

        to\_delete = nullptr;

    }

    this->\_free\_blocks\_list = nullptr;

}

typename Allocator::void\_pointer ListAllocator::alloc(size\_type new\_block\_size)

{

    BlockHeader\* prev\_block = nullptr;

    BlockHeader\* cur\_block = this->\_free\_blocks\_list;

    size\_type adjusted\_size = new\_block\_size + sizeof(BlockHeader);

    while (cur\_block) {

        if (cur\_block->\_size >= adjusted\_size) {

            if (cur\_block->\_size >= adjusted\_size + sizeof(BlockHeader)) {

                BlockHeader\* new\_block = reinterpret\_cast<BlockHeader\*>(reinterpret\_cast<int8\_t\*>(cur\_block) + adjusted\_size);

                new\_block->\_size = cur\_block->\_size - adjusted\_size - sizeof(BlockHeader);

                new\_block->\_next = cur\_block->\_next;

                cur\_block->\_next = new\_block;

                cur\_block->\_size = adjusted\_size;

            }

            if (prev\_block) {

                prev\_block->\_next = cur\_block->\_next;

            } else {

                this->\_free\_blocks\_list = cur\_block->\_next;

            }

            return reinterpret\_cast<int8\_t\*>(cur\_block) + sizeof(BlockHeader);

        }

        prev\_block = cur\_block;

        cur\_block = cur\_block->\_next;

    }

    return nullptr;

}

void ListAllocator::free(void\_pointer block)

{

    if (block == nullptr) return;

    BlockHeader\* header = reinterpret\_cast<BlockHeader\*>(static\_cast<int8\_t\*>(block) - sizeof(BlockHeader));

    header->\_next = this->\_free\_blocks\_list;

    this->\_free\_blocks\_list = header;

}

./srcallocator/ MacKuseyCarelsAllocator.cpp

#include "../../include/allocator/MacKuseyCarelsAllocator.h"

using namespace allocator;

MacKuseyCarelsAllocator::MacKuseyCarelsAllocator(void\_pointer real\_memory, size\_type memory\_size)

{

    \_memory = reinterpret\_cast<void\*>(reinterpret\_cast<int8\_t\*>(real\_memory) + sizeof(MacKuseyCarelsAllocator));

    \_free\_pages\_list = nullptr;

    \_memory\_size = memory\_size - sizeof(MacKuseyCarelsAllocator);

    \_page\_size = getpagesize();

}

MacKuseyCarelsAllocator::~MacKuseyCarelsAllocator()

{

    Page\* cur\_page = this->\_free\_pages\_list;

    while (cur\_page) {

        Page\* to\_delete = cur\_page;

        cur\_page = cur\_page->\_next;

        munmap(to\_delete, \_page\_size);

        to\_delete = nullptr;

    }

    \_free\_pages\_list = nullptr;

}

typename Allocator::void\_pointer MacKuseyCarelsAllocator::alloc(size\_type new\_block\_size)

{

    if (\_memory\_size < new\_block\_size) return nullptr;

    size\_t rounded\_block\_size = 1;

    while (rounded\_block\_size < new\_block\_size) {

        rounded\_block\_size \*= 2;

    }

    Page\* prev\_page = nullptr;

    Page\* cur\_page = \_free\_pages\_list;

    while (cur\_page) {

        if (!cur\_page->\_is\_large && cur\_page->\_block\_size == rounded\_block\_size) {

            void\_pointer block = reinterpret\_cast<void\_pointer>(cur\_page);

            \_free\_pages\_list = cur\_page->\_next;

            \_memory\_size -= new\_block\_size;

            return block;

        }

        prev\_page = cur\_page;

        cur\_page = cur\_page->\_next;

    }

    if (\_memory\_size < \_page\_size) return nullptr;

    Page\* new\_page = reinterpret\_cast<Page\*>(mmap(NULL, \_page\_size,

                                                  PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS,

                                                  -1, 0));

    if (new\_page == MAP\_FAILED) {

        throw std::bad\_alloc();

    }

    new\_page->\_is\_large = false;

    new\_page->\_block\_size = rounded\_block\_size;

    new\_page->\_next = nullptr;

    size\_t num\_blocks = \_page\_size / rounded\_block\_size;

    for (size\_t i = 0; i != num\_blocks; ++i) {

        Page\* block\_page = reinterpret\_cast<Page\*>(reinterpret\_cast<int8\_t\*>(new\_page) + i \* rounded\_block\_size);

        block\_page->\_is\_large = false;

        block\_page->\_block\_size = rounded\_block\_size;

        block\_page->\_next = this->\_free\_pages\_list;

        this->\_free\_pages\_list = block\_page;

    }

    void\_pointer block = reinterpret\_cast<void\_pointer>(new\_page);

    this->\_free\_pages\_list = new\_page->\_next;

    return block;

}

void MacKuseyCarelsAllocator::free(void\_pointer block)

{

    if (block == nullptr) return;

    Page\* page = reinterpret\_cast<Page\*>(block);

    page->\_next = \_free\_pages\_list;

    \_free\_pages\_list = page;

}

./src/benchmark/benchmark.cpp

#include <chrono>

#include <cstdlib>

#include <vector>

#include "../include/allocator/ListAllocator.h"

#include "../include/allocator/MacKuseyCarelsAllocator.h"

using namespace allocator;

size\_t page\_size = sysconf(\_SC\_PAGESIZE);

void benchmark() {

    void\* list\_memory = sbrk(10000 \* page\_size);

    void\* MKC\_memory = mmap(NULL, 1000 \* page\_size, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);

    ListAllocator list\_alloc(list\_memory, 10000 \* page\_size);

    MacKuseyCarelsAllocator MKC\_alloc(MKC\_memory, 1000 \* page\_size);

    std::vector<void\*> list\_blocks;

    std::vector<void\*> MKC\_blocks;

    std::cout << "Comparing ListAllocator and MacKuseyCarelsAllocator" << std::endl;

    std::cout << "Block allocation rate" << std::endl;

    auto start\_time = std::chrono::steady\_clock::now();

    for (size\_t i = 0; i != 100000; ++i) {

        void\* block = list\_alloc.alloc(i % 50 + 10);

        list\_blocks.push\_back(block);

    }

    auto end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of alloc ListAllocator: " <<

                  std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() <<

                  " milliseconds" << std::endl;

    start\_time = std::chrono::steady\_clock::now();

    for (size\_t i = 0; i != 100000; ++i) {

        void\* block = MKC\_alloc.alloc(i % 50 + 10);

        MKC\_blocks.push\_back(block);

    }

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of alloc MacKuseyCarelsAllocator: " <<

                  std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() <<

                  " milliseconds" << std::endl;

    std::cout << "Block free rate" << std::endl;

    start\_time = std::chrono::steady\_clock::now();

    for (size\_t i = 0; i != list\_blocks.size(); ++i) {

        list\_alloc.free(list\_blocks[i]);

        if (i < 20) {

            std::cout << list\_blocks[i] << std::endl;

        }

    }

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of free ListAllocator: " <<

                  std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() <<

                  " milliseconds" << std::endl;

    start\_time = std::chrono::steady\_clock::now();

    for (size\_t i = 0; i != MKC\_blocks.size(); ++i) {

        MKC\_alloc.free(MKC\_blocks[i]);

        if (i < 20) {

            std::cout << MKC\_blocks[i] << std::endl;

        }

    }

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of free MacKuseyCarelsAllocator: " <<

                  std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() <<

                  " milliseconds" << std::endl;

}

./run.cpp

#include "include/allocator/ListAllocator.h"

#include "include/allocator/MacKuseyCarelsAllocator.h"

#include "src/benchmarks/benchmark.cpp"

using namespace allocator;

int main() {

    size\_t page\_size = sysconf(\_SC\_PAGESIZE);

    void\* list\_memory = sbrk(10 \* page\_size);

    ListAllocator list\_alloc(list\_memory, 10 \* page\_size);

    void\* MKC\_memory = mmap(NULL, 10 \* page\_size, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);

    MacKuseyCarelsAllocator MKC\_alloc(MKC\_memory, 10 \* page\_size);

    void\* block1 = list\_alloc.alloc(10000);

    void\* block2 = list\_alloc.alloc(10);

    void\* block3 = list\_alloc.alloc(44651047871);

    void\* block4 = MKC\_alloc.alloc(1024);

    void\* block5 = MKC\_alloc.alloc(2049);

    void\* block6 = MKC\_alloc.alloc(144420166);

    printf("Block 1: %p\n", block1);

    printf("Block 2: %p\n", block2);

    printf("Block 3: %p\n", block3);

    printf("Block 4: %p\n", block4);

    printf("Block 5: %p\n", block5);

    printf("Block 6: %p\n", block6);

    benchmark();

    return 0;

}

./CMakeLists.txt

cmake\_minimum\_required(VERSION 3.10)

project(KP)

add\_compile\_options(-Wall -g -O0)

set(CMAKE\_CXX\_STANDARD 20)

set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)

set(INCLUDE\_DIR ${CMAKE\_CURRENT\_SOURCE\_DIR}/include)

set(SOURCE\_DIR ${CMAKE\_CURRENT\_SOURCE\_DIR}/src)

include\_directories(${INCLUDE\_DIR})

file(GLOB\_RECURSE SOURCES LIST\_DIRECTORIES false ${SOURCE\_DIR}/\*.cpp)

file(GLOB\_RECURSE INCLUDES LIST\_DIRECTORIES false ${INCLUDE\_DIR}/\*.hpp)

add\_library(

            ${CMAKE\_PROJECT\_NAME}\_lib

            ${SOURCES}

            ${INCLUDES}

            ${ENUMS}

           )

add\_executable(

               ${CMAKE\_PROJECT\_NAME}\_exe

               ${CMAKE\_CURRENT\_SOURCE\_DIR}/run.cpp

              )

set\_target\_properties(

                      ${PROJECT\_NAME}\_exe PROPERTIES

                      CXX\_STANDARD 20

                      CXX\_STANDARD\_REQUIRED YES

                      CXX\_EXTENSIONS NO

                     )

add\_dependencies(${CMAKE\_PROJECT\_NAME}\_exe ${CMAKE\_PROJECT\_NAME}\_lib)

target\_link\_libraries(${CMAKE\_PROJECT\_NAME}\_exe ${CMAKE\_PROJECT\_NAME}\_lib)

./Makefile

.PHONY: build run clean strace

build: clean

    mkdir build

    cd ./build; cmake ..; make all

run:

    ./build/\*\_exe

clean:

    rm -rf ./build/

strace:

    strace -f ./build/\*\_exe

## Использование утилиты strace

Без бенчмарка, потому что слишком много системных вызовов на выделение и очищение памяти

execve("./build/KP\_exe", ["./build/KP\_exe"], 0x7ffc96f74f28 /\* 60 vars \*/) = 0

brk(NULL) = 0x5622f08d3000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffd746cef80) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f7c4b098000

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=67103, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 67103, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f7c4b087000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libstdc++.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=2260296, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 2275520, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f7c4ae00000

mprotect(0x7f7c4ae9a000, 1576960, PROT\_NONE) = 0

mmap(0x7f7c4ae9a000, 1118208, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x9a000) = 0x7f7c4ae9a000

mmap(0x7f7c4afab000, 454656, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ab000) = 0x7f7c4afab000

mmap(0x7f7c4b01b000, 57344, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x21a000) = 0x7f7c4b01b000

mmap(0x7f7c4b029000, 10432, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f7c4b029000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libgcc\_s.so.1", 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|0644, st\_size=125488, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 127720, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f7c4b067000

mmap(0x7f7c4b06a000, 94208, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x3000) = 0x7f7c4b06a000

mmap(0x7f7c4b081000, 16384, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1a000) = 0x7f7c4b081000

mmap(0x7f7c4b085000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1d000) = 0x7f7c4b085000

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\0 =\340\2563\265?\356\25x\261\27\313A#\350"..., 68, 896) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2216304, ...}, 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, 2260560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f7c4aa00000

mmap(0x7f7c4aa28000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f7c4aa28000

mmap(0x7f7c4abbd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f7c4abbd000

mmap(0x7f7c4ac15000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x214000) = 0x7f7c4ac15000

mmap(0x7f7c4ac1b000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f7c4ac1b000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libm.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=940560, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 942344, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f7c4ad19000

mmap(0x7f7c4ad27000, 507904, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xe000) = 0x7f7c4ad27000

mmap(0x7f7c4ada3000, 372736, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x8a000) = 0x7f7c4ada3000

mmap(0x7f7c4adfe000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xe4000) = 0x7f7c4adfe000

close(3) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f7c4b065000

arch\_prctl(ARCH\_SET\_FS, 0x7f7c4b0663c0) = 0

set\_tid\_address(0x7f7c4b066690) = 5489

set\_robust\_list(0x7f7c4b0666a0, 24) = 0

rseq(0x7f7c4b066d60, 0x20, 0, 0x53053053) = 0

mprotect(0x7f7c4ac15000, 16384, PROT\_READ) = 0

mprotect(0x7f7c4adfe000, 4096, PROT\_READ) = 0

mprotect(0x7f7c4b085000, 4096, PROT\_READ) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f7c4b063000

mprotect(0x7f7c4b01b000, 45056, PROT\_READ) = 0

mprotect(0x5622eeef8000, 4096, PROT\_READ) = 0

mprotect(0x7f7c4b0d2000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7f7c4b087000, 67103) = 0

getrandom("\xb9\xae\x29\x78\xef\x33\xb2\x70", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x5622f08d3000

brk(0x5622f08f4000) = 0x5622f08f4000

futex(0x7f7c4b02977c, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

brk(0x5622f08fe000) = 0x5622f08fe000

mmap(NULL, 40960, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f7c4b08e000

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f7c4b0d1000

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f7c4b08d000

newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

write(1, "Block 1: 0x5622f08f4020\n", 24Block 1: 0x5622f08f4020

) = 24

write(1, "Block 2: 0x5622f08f6740\n", 24Block 2: 0x5622f08f6740

) = 24

write(1, "Block 3: (nil)\n", 15Block 3: (nil)

) = 15

write(1, "Block 4: 0x7f7c4b0d1000\n", 24Block 4: 0x7f7c4b0d1000

) = 24

write(1, "Block 5: 0x7f7c4b08d000\n", 24Block 5: 0x7f7c4b08d000

) = 24

write(1, "Block 6: (nil)\n", 15Block 6: (nil)

) = 15

exit\_group(0) = ?

+++ exited with 0 +++

## Демонстрация работы программы

baronpipistron@BaronPIpistron:~/MAI\_OS/KP$ make run

./build/\*\_exe

Block 1: 0x55b6ed9d0020

Block 2: 0x55b6ed9d2740

Block 3: (nil)

Block 4: 0x7f8d5db8b000

Block 5: 0x7f8d5db47000

Block 6: (nil)

Comparing ListAllocator and MacKuseyCarelsAllocator

Block allocation rate

Time of alloc ListAllocator: 6 milliseconds

Time of alloc MacKuseyCarelsAllocator: 477 milliseconds

Block free rate

0x55b6ed9da020

0x55b6ed9da03a

0x55b6ed9da055

0x55b6ed9da071

0x55b6ed9da08e

0x55b6ed9da0ac

0x55b6ed9da0cb

0x55b6ed9da0eb

0x55b6ed9da10c

0x55b6ed9da12e

0x55b6ed9da151

0x55b6ed9da175

0x55b6ed9da19a

0x55b6ed9da1c0

0x55b6ed9da1e7

0x55b6ed9da20f

0x55b6ed9da238

0x55b6ed9da262

0x55b6ed9da28d

0x55b6ed9da2b9

Time of free ListAllocator: 0 milliseconds

0x7f8d5db46000

0x7f8d5db45000

0x7f8d5db44000

0x7f8d5db43000

0x7f8d5db42000

0x7f8d5db41000

0x7f8d5da35000

0x7f8d5da34000

0x7f8d5da33000

0x7f8d5da32000

0x7f8d5da31000

0x7f8d5da30000

0x7f8d5da2f000

0x7f8d5da2e000

0x7f8d5da2d000

0x7f8d5da2c000

0x7f8d5d7ff000

0x7f8d5d7fe000

0x7f8d5d7fd000

0x7f8d5d7fc000

Time of free MacKuseyCarelsAllocator: 2 milliseconds

## Вывод

В ходе выполнения данной работы, я сравнил два алгоритма аллокации «Списки свободных блоков (первое подходящее)» и «Мак-Кьюзи-Кэрелса». Работа оказалась очень интересной и выполнять ее было одно удовольствие. Пришлось погрузиться в глубины ОС Linux чтобы правильно написать оба алгоритма и понять как сама ОС взаимодействует с памятью на уровне ядра. Узнал очень много нового про саму ОС и аллокаторы, а также усовершенствовал свои навыки во владении С++.

Результат сравнения двух алгоритмов аллокации оказался довольно интересный. «Списки свободных блоков (первое подходящее)» оказался в разы быстрее «Мак-Кьюзи-Кэрелса». Я считаю, что это связано с тем, что второй алгоритм стучится в ядро, если не находит страницы с подходящим размером блоков, а вызовы в ядро очень долгие. Но с другой стороны «Мак-Кьюзи-Кэрелса» предлагает нам более надежное хранение наших данных и исключает фрагментацию, поиск свободных блоков не занимает слишком много времени, и в целом нахожу его более универсальным, в то время как поиск в первом алгоритме будет увеличиваться пропорционально тому, чем дальше в списке находится подходящий блок. Освобождают память оба алгоритма одинаково хорошо.

В целом работой остался доволен, выполнять было крайне интересно. Но буду пользоваться скорее всего все равно моим любимым стандартным std::allocator<T> и не знать никаких проблем. Несомненно, есть шанс, что понадобится написать свой аллокатор, но он крайне мал.

Работа на 8 из 10 – долго мучался со вторым алгоритмом.