Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект по курсу
«Операционные системы»

Студент: Ворошилов Кирилл Сергеевич
Группа: М8О-201Б-21
Вариант: 11
Преподаватель Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/kiviyi/operation-system/tree/cp

Постановка задачи

Цель работы

1. Приобретение практических навыков в использовании знаний, полученных в течении курса
2. Проведение исследования в выбранной предметной области

Постановка задачи

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

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.

Вариант №17: Необходимо сравнить два алгоритма аллокации: алгоритм Мак-Кьюзи-Кэрелса и алгоритм двойников.

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).

Общие сведения о программе

Программа представляет из себя 6 файлов: main.cpp mck-k.hpp mck-k.cpp

BuddyAllocator.hpp BuddyAllocator.cpp CMakeLists.txt

Описание алгоритма аллокации

Алгоритм двойников

В данном алгоритме свободная часть памяти разбивается до тех пор, пока не выйдет блок памяти нужного размера, в каждом блоке есть идентификатор, обозначающий занят или свободен блок. Если освобождается блок и его двойник оказывается свободен, то двойников сливают. Полученный блок пытаются слить с его двойником. Блок, который не удалось слить добавляют в список свободных блоков. Свободные блоки хранятся в двусвязном списке.

Алгоритм Мак-Кьюзи-Кэрелса

Данный алгоритм является улучшенной версией алгоритма блоков степени 2, в нём свободные блоки хранятся в списках, но размер блока хранится на уровне самого аллокатора в специальном массиве.

Исходный код

main.cpp

```
#include <iostream>
#include <chrono>

#include "BuddyAllocator.hpp"
#include "mck-k.hpp"

int main()
{
   using namespace std::chrono;

   std::cout << "First test, allocator initialization:" << std::endl;

   int pagesAmount = 10;
   std::vector<int> pagesFragments = {32, 128, 256, 1024, 512, 256, 256, 1024, 16, 256};

   steady_clock::time_point mcKKAllocatorInitStart = steady_clock::now();
```

```
    McKKAllocator mcKKAllocator(pagesAmount, pagesFragments);
    steady_clock::time_point mcKKAllocatorInitEnd = steady_clock::now();

    std::cout << "McKusick-Karels: " <<
std::chrono::duration_cast<std::chrono::nanoseconds>(mcKKAllocatorInitEnd -
mcKKAllocatorInitStart).count() << " ns" << std::endl;

    steady_clock::time_point buddyAllocatorInitStart = steady_clock::now();
    BuddyAllocator buddyAllocator(4096);
    steady_clock::time_point buddyAllocatorInitEnd = steady_clock::now();

    std::cout << "Buddy: " <<
std::chrono::duration_cast<std::chrono::nanoseconds>(buddyAllocatorInitEnd -
buddyAllocatorInitStart).count() << " ns";
    std::cout << std::endl;

    std::cout << std::endl;

    std::cout << "Second test, allocate 10 char[256], deallocate 5 of them, allocate 5 char[128]:\n";

    std::vector<char *> pointers2(10, 0);
    steady_clock::time_point mcKKTest1Start = steady_clock::now();
    for (int i = 0; i < 10; ++i){
        pointers2[i] = (char *)mcKKAllocator.allocate(256);
    }
    for (int i = 5; i < 10; ++i){
        mcKKAllocator.deallocate(pointers2[i]);
    }
    for (int i = 5; i < 10; ++i){
        pointers2[i] = (char *)mcKKAllocator.allocate(128);
    }
    steady_clock::time_point mcKKTest1End = steady_clock::now();
    std::cerr << "McKusick-Karels: " <<
std::chrono::duration_cast<std::chrono::microseconds>(mcKKTest1End - mcKKTest1Start).count() << "
microseconds" << std::endl;
```

```
    BuddyAllocator allocator(8192);
    std::vector<char *> pointers(1000, 0);
    steady_clock::time_point buddyTestStart = steady_clock::now();
    for (int i = 0; i < 10; ++i){
        pointers[i] = (char *)allocator.allocate(256);
    }
    for (int i = 5; i < 10; ++i){
        allocator.deallocate(pointers[i]);
    }
    for (int i = 5; i < 10; ++i){
        pointers[i] = (char *)allocator.allocate(128);
    }
    steady_clock::time_point buddyTestEnd = steady_clock::now();
    std::cerr << "Buddy: " << std::chrono::duration_cast<std::chrono::microseconds>(buddyTestEnd -
buddyTestStart).count() << " microseconds" << std::endl;
    for(int i = 0; i < 10; ++i){
        allocator.deallocate(pointers[i]);
    }

    std::cout << std::endl;

    for (int i = 0; i < 10; ++i){
        mcKKAllocator.deallocate(pointers2[i]);
    }

    std::cerr << "Third test, Allocate and free 15 20 bytes arrays:\n";
    pagesAmount = 15;
    std::vector<int> pagesFragment2 = {1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024,
1024, 1024, 1024, 1024, 1024};

    McKKAllocator allocator2(pagesAmount, pagesFragment2);

    std::vector<char *> pointer2(15, 0);
    steady_clock::time_point alloc_start = steady_clock::now();
    for (int i = 0; i < 15; ++i){
```

```
        pointer2[i] = (char *)allocator2.allocate(20);
    }
    steady_clock::time_point alloc_end = steady_clock::now();
    for (int i = 0; i < 15; ++i){
        allocator2.deallocate(pointer2[i]);
    }
    steady_clock::time_point test_end = steady_clock::now();
    std::cerr << "McKusick-Karels allocation: " << duration_cast<std::chrono::microseconds>(alloc_end -
alloc_start).count() << " microseconds"<< "\n"
              << "McKusick-Karels deallocation: " << duration_cast<std::chrono::microseconds>(test_end -
alloc_end).count() << " microseconds "<< "\n";

    std::cout << std::endl;

    BuddyAllocator allocator3(32000);
    std::vector<char *> pointers3(150, 0);
    alloc_start = steady_clock::now();
    for (int i = 0; i < 15; ++i){
        pointers3[i] = (char *)allocator3.allocate(20);
    }
    alloc_end = steady_clock::now();
    for (int i = 0; i < 15; ++i){
        allocator3.deallocate(pointers3[i]);
    }
    test_end = steady_clock::now();
    std::cerr << "Buddy allocation: " << duration_cast<std::chrono::microseconds>(alloc_end -
alloc_start).count() << " microseconds"<< "\n"
              << "Buddy deallocation: " << duration_cast<std::chrono::microseconds>(test_end -
alloc_end).count() << " microseconds "<< "\n ";

        return 0;


}
```

mck-k.cpp

```
#include <iostream>
```

```
#include <chrono>

#include "BuddyAllocator.hpp"
#include "mck-k.hpp"

int main()
{
  using namespace std::chrono;

  std::cout << "First test, allocator initialization:" << std::endl;

  int pagesAmount = 10;
  std::vector<int> pagesFragments = {32, 128, 256, 1024, 512, 256, 256, 1024, 16,
256};

  steady_clock::time_point mcKKAllocatorInitStart = steady_clock::now();
  McKKAllocator mcKKAllocator(pagesAmount, pagesFragments);
  steady_clock::time_point mcKKAllocatorInitEnd = steady_clock::now();

  std::cout << "McKusick-Karels: " <<
std::chrono::duration_cast<std::chrono::nanoseconds>(mcKKAllocatorInitEnd -
mcKKAllocatorInitStart).count() << " ns" << std::endl;

  steady_clock::time_point buddyAllocatorInitStart = steady_clock::now();
  BuddyAllocator buddyAllocator(4096);
  steady_clock::time_point buddyAllocatorInitEnd = steady_clock::now();

  std::cout << "Buddy: " <<
std::chrono::duration_cast<std::chrono::nanoseconds>(buddyAllocatorInitEnd -
buddyAllocatorInitStart).count() << " ns";
  std::cout << std::endl;

  std::cout << std::endl;

  std::cout << "Second test, allocate 10 char[256], deallocate 5 of them,
allocate 5 char[128]:\n";

  std::vector<char *> pointers2(10, 0);
  steady_clock::time_point mcKKTest1Start = steady_clock::now();
  for (int i = 0; i < 10; ++i){
    pointers2[i] = (char *)mcKKAllocator.allocate(256);
  }
  for (int i = 5; i < 10; ++i){
    mcKKAllocator.deallocate(pointers2[i]);
  }
  for (int i = 5; i < 10; ++i){
    pointers2[i] = (char *)mcKKAllocator.allocate(128);
  }
  steady_clock::time_point mcKKTest1End = steady_clock::now();
  std::cerr << "McKusick-Karels: " <<
```

```
std::chrono::duration_cast<std::chrono::microseconds>(mcKKTest1End -
mcKKTest1Start).count() << " microseconds" << std::endl;


  BuddyAllocator allocator(8192);
  std::vector<char *> pointers(1000, 0);
  steady_clock::time_point buddyTestStart = steady_clock::now();
  for (int i = 0; i < 10; ++i){
    pointers[i] = (char *)allocator.allocate(256);
  }
  for (int i = 5; i < 10; ++i){
    allocator.deallocate(pointers[i]);
  }
  for (int i = 5; i < 10; ++i){
    pointers[i] = (char *)allocator.allocate(128);
  }
  steady_clock::time_point buddyTestEnd = steady_clock::now();
  std::cerr << "Buddy: " <<
std::chrono::duration_cast<std::chrono::microseconds>(buddyTestEnd -
buddyTestStart).count() << " microseconds" << std::endl;
  for(int i = 0; i < 10; ++i){
    allocator.deallocate(pointers[i]);
  }

  std::cout << std::endl;

  for (int i = 0; i < 10; ++i){
    mcKKAllocator.deallocate(pointers2[i]);
  }

  std::cerr << "Third test, Allocate and free 15 20 bytes arrays:\n";
  pagesAmount = 15;
  std::vector<int> pagesFragment2 = {1024, 1024, 1024, 1024, 1024, 1024, 1024,
1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024};

  McKKAllocator allocator2(pagesAmount, pagesFragment2);

  std::vector<char *> pointer2(15, 0);
  steady_clock::time_point alloc_start = steady_clock::now();
  for (int i = 0; i < 15; ++i){
    pointer2[i] = (char *)allocator2.allocate(20);
  }
  steady_clock::time_point alloc_end = steady_clock::now();
  for (int i = 0; i < 15; ++i){
    allocator2.deallocate(pointer2[i]);
  }
  steady_clock::time_point test_end = steady_clock::now();
  std::cerr << "McKusick-Karels allocation: " <<
duration_cast<std::chrono::microseconds>(alloc_end - alloc_start).count() << "
microseconds"<< "\n"
```

```
      << "McKusick-Karels deallocation: " <<
duration_cast<std::chrono::microseconds>(test_end - alloc_end).count() << "
microseconds "<< "\n";

  std::cout << std::endl;

  BuddyAllocator allocator3(32000);
  std::vector<char *> pointers3(150, 0);
  alloc_start = steady_clock::now();
  for (int i = 0; i < 15; ++i){
    pointers3[i] = (char *)allocator3.allocate(20);
  }
  alloc_end = steady_clock::now();
  for (int i = 0; i < 15; ++i){
    allocator3.deallocate(pointers3[i]);
  }
  test_end = steady_clock::now();
  std::cerr << "Buddy allocation: " <<
duration_cast<std::chrono::microseconds>(alloc_end - alloc_start).count() << "
microseconds"<< "\n"
      << "Buddy deallocation: " <<
duration_cast<std::chrono::microseconds>(test_end - alloc_end).count() << "
microseconds"<< "\n";

  std::cout << std::endl;

  std::cout << "Fout test, allocate 100 char[128], deallocate 100 of them,
allocate 100 char[256]:\n";

  int pagesmout = 100;
  std::vector<int> pageragmens;

  for(int i = 0; i < 100; ++i){
    pageragmens.push_back(1024);
  }


  McKKAllocator cKKAllocatr(pagesmout, pageragmens);


  std::vector<char *> pointers5(100, 0);

  steady_clock::time_point mcKKTest5Start = steady_clock::now();
  for (int i = 0; i < 100; ++i){
    pointers5[i] = (char *)cKKAllocatr.allocate(128);
  }
  for (int i = 0; i < 100; ++i){
    cKKAllocatr.deallocate(pointers5[i]);
  }
```

```
  for (int i = 0; i < 100; ++i){
    pointers5[i] = (char *)cKKAllocatr.allocate(256);
  }
  steady_clock::time_point mcKKTest5End = steady_clock::now();
  std::cerr << "McKusick-Karels: " <<
std::chrono::duration_cast<std::chrono::microseconds>(mcKKTest5End -
mcKKTest5Start).count() << " microseconds" << std::endl;


  BuddyAllocator allocator5(1048576);
  std::vector<char *> pointers5_b(100, 0);

  steady_clock::time_point buddyTest5Start = steady_clock::now();
  for (int i = 0; i < 100; ++i){
    pointers5_b[i] = (char *)allocator5.allocate(120);
  }

  for (int i = 0; i < 100; ++i){
    allocator5.deallocate(pointers5_b[i]);
  }

  for (int i = 0; i < 100; ++i){
    pointers5_b[i] = (char *)allocator5.allocate(256);
  }
  steady_clock::time_point buddyTest5End = steady_clock::now();

  std::cerr << "Buddy: " <<
std::chrono::duration_cast<std::chrono::microseconds>(buddyTest5End -
buddyTest5Start).count() << " microseconds" << std::endl;
  for(int i = 0; i < 100; ++i){
    allocator5.deallocate(pointers5_b[i]);
  }
  std::cout << std::endl;

  for (int i = 0; i < 100; ++i){
    cKKAllocatr.deallocate(pointers5[i]);
  }

  std::cout << "Five test, allocate 100 char[20], deallocate 100 of them,
allocate 100 char[40]:\n";

  int pagesmout6 = 100;
  std::vector<int> pageragmens6;

  for(int i = 0; i < 100; ++i){
    pageragmens6.push_back(1024);
  }
```

```
  McKKAllocator cKKAllocatr6(pagesmout6, pageragmens6);


  std::vector<char *> pointers6(100, 0);

  steady_clock::time_point mcKKTest6Start = steady_clock::now();
  for (int i = 0; i < 100; ++i){
    pointers6[i] = (char *)cKKAllocatr6.allocate(20);
  }
  for (int i = 0; i < 100; ++i){
    cKKAllocatr6.deallocate(pointers6[i]);
  }
  for (int i = 0; i < 100; ++i){
    pointers6[i] = (char *)cKKAllocatr6.allocate(40);
  }
  steady_clock::time_point mcKKTest6End = steady_clock::now();
  std::cerr << "McKusick-Karels: " <<
std::chrono::duration_cast<std::chrono::microseconds>(mcKKTest6End -
mcKKTest6Start).count() << " microseconds" << std::endl;


  BuddyAllocator allocator6(1048576);
  std::vector<char *> pointers6_b(100, 0);

  steady_clock::time_point buddyTest6Start = steady_clock::now();
  for (int i = 0; i < 100; ++i){
    pointers6_b[i] = (char *)allocator6.allocate(20);
  }

  for (int i = 0; i < 100; ++i){
    allocator6.deallocate(pointers6_b[i]);
  }

  for (int i = 0; i < 100; ++i){
    pointers6_b[i] = (char *)allocator6.allocate(40);
  }
  steady_clock::time_point buddyTest6End = steady_clock::now();

  std::cerr << "Buddy: " <<
std::chrono::duration_cast<std::chrono::microseconds>(buddyTest6End -
buddyTest6Start).count() << " microseconds" << std::endl;
  for(int i = 0; i < 100; ++i){
    allocator6.deallocate(pointers6_b[i]);
  }
  std::cout << std::endl;

  for (int i = 0; i < 100; ++i){
    cKKAllocatr6.deallocate(pointers6[i]);
  }
```

```
    return 0;

}
```

mck-k.hpp

```
#include <iostream>
#include <list>
#include <algorithm>
#include <vector>
#include <map>

#define PAGE_SIZE 1024

struct Page{
    int blockSize;
    char* start;
    char* end;
};

class McKKAllocator {
private:
    std::vector<int> powsOf2 = {16,32,64,128,256,512,1024};
    std::vector<std::list<char*>> freeBlocksLists;
    std::vector<Page> kMemSize;
    char* data;

public:
    McKKAllocator(int pagesAmount, std::vector<int>& pagesFragments); // pagesFragments is a vector with sizes of blocks
                                                                      // on which page is splitted
    void* allocate(int bytesAmount);
    void deallocate(void *ptr);

    ~McKKAllocator();

};
```

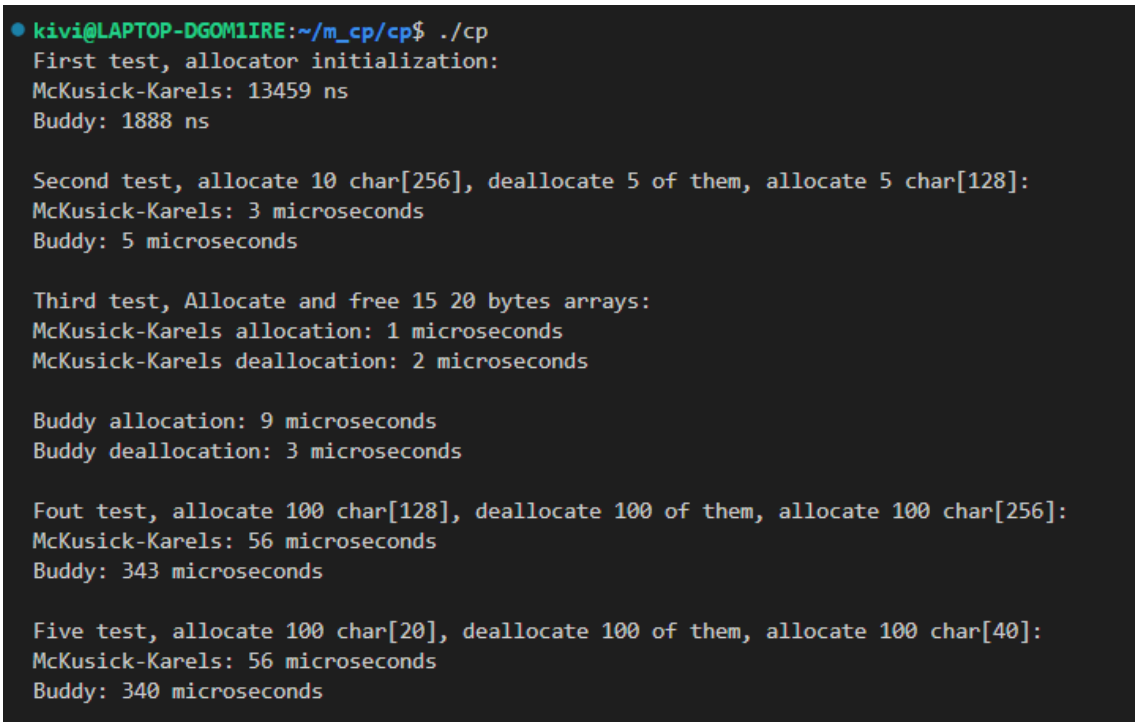
BuddyAllocator.cpp

```
#include "BuddyAllocator.hpp"
#include <algorithm>
void setBlock(char *p, size_t size){
   *((int *)p) = size;
}

int getSize(char *p){
   return *((int *)p);
}

BuddyAllocator::BuddyAllocator(const size_t allowedSize) : mem_size{allowedSize}{
   data = (char *)malloc(allowedSize);
   setBlock(data, allowedSize);
   freeBlocks.push_back(data);
}

void *BuddyAllocator::allocate(size_t mem_size){
   if (mem_size == 0)    {
      return nullptr;
   }

   int index = -1;
   mem_size += sizeof(int);
   for (int i = 0; i < freeBlocks.size(); ++i){
      if (getSize(freeBlocks[i]) >= mem_size)
      {
         index = i;
         break;
      }
   }
   if (index == -1){
      std::cout << "There isn't memory\n";
   }

   size_t currentBlockSize = getSize(freeBlocks[index]);
```

```
    while ((currentBlockSize % 2 == 0) && (currentBlockSize / 2 >= mem_size)){
        currentBlockSize /= 2;
        char *newBlock = freeBlocks[index] + currentBlockSize;
        setBlock(newBlock, currentBlockSize);
        freeBlocks.push_back(newBlock);
    }

    setBlock(freeBlocks[index], currentBlockSize);
    freeBlocks.erase(std::next(freeBlocks.begin(), index));
    return freeBlocks[index] + sizeof(int);
}

void BuddyAllocator::deallocate(void *ptr){
    char *c_ptr = (char *)ptr - sizeof(int);
    size_t size = getSize(c_ptr);
    auto found = std::find(freeBlocks.begin(), freeBlocks.end(), c_ptr + size);
    if (found != freeBlocks.end()){
        freeBlocks.erase(found);
        setBlock(c_ptr, size * 2);
        freeBlocks.push_back(c_ptr);
        return;
    }

    found = std::find(freeBlocks.begin(), freeBlocks.end(), c_ptr - size);
    if (found != freeBlocks.end()){
        setBlock(c_ptr - size, size * 2);
        return;
    }

    freeBlocks.push_back(c_ptr);
}

BuddyAllocator::~BuddyAllocator(){
    free(data);
}
```

```
#ifndef BUDDYALLOCATOR_H
#define BUDDYALLOCATOR_H
#include <vector>
#include <iostream>
class BuddyAllocator
{
public:
   BuddyAllocator(const size_t allowedSize);
   ~BuddyAllocator();
   void *allocate(size_t mem_size);
   void deallocate(void *ptr);

private:
   std::vector<char *> freeBlocks;
   char *data;
   size_t mem_size;
};
#endif // BUDDYALLOCATOR_H
```

Демонстрация работы программы



Анализ полученных результатов

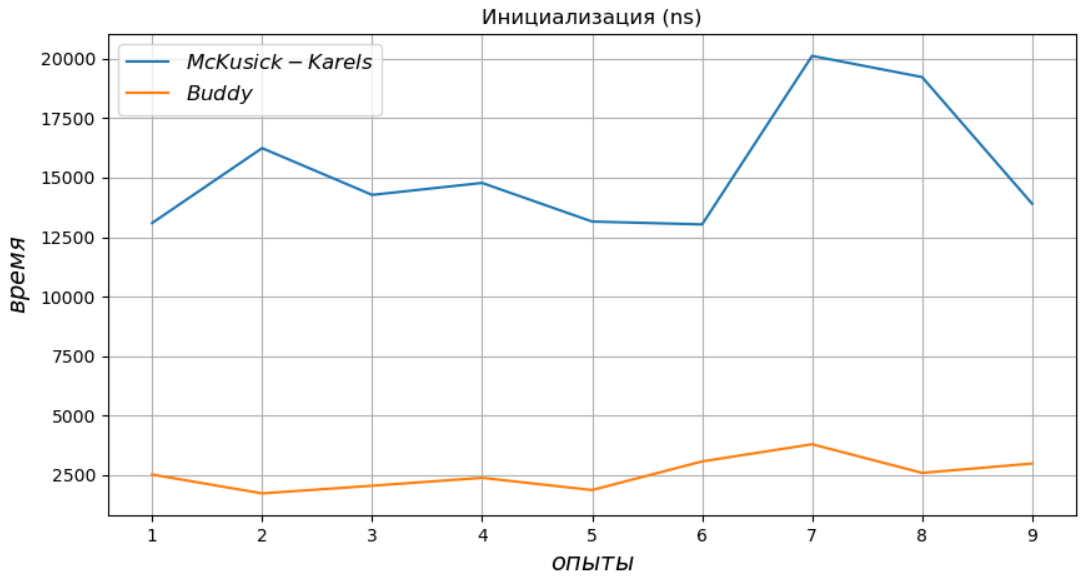
Первый тест проверяет, сколько времени требуется для инициализации — аллгоритм двойников инициализируется гораздо быстрее, так как алгоритму Мак-Кьюзи-Кэрелса требуется время для создания списков.

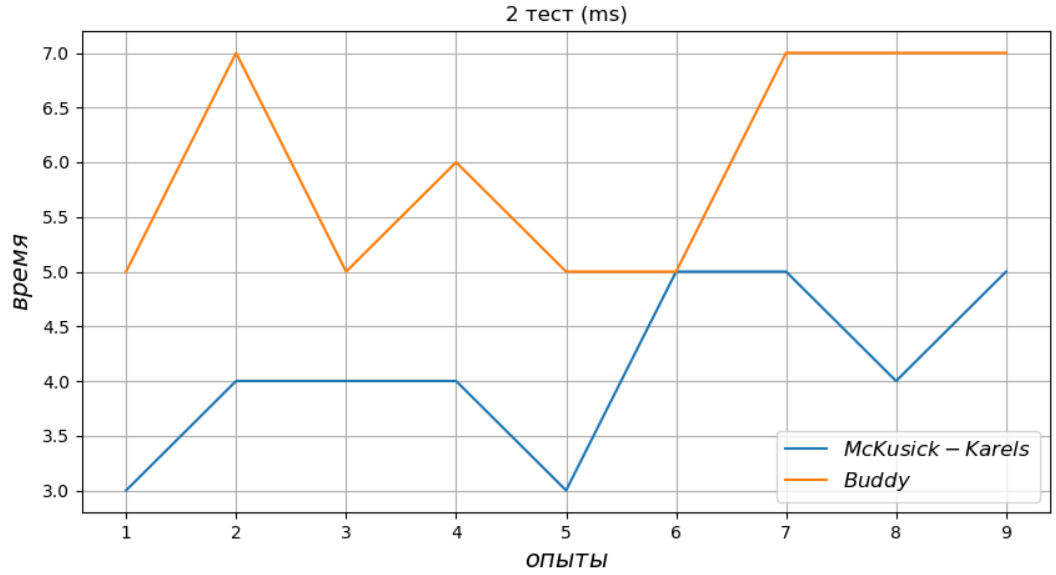
Второй тест проверяет работу алгоритмов для малого количества указателей. Из результатов видно, что алгоритм двойников работает медленнее, чем алгоритм Мак-Кьюзи-Кэрелса, однако разница не достигает 2 раз, что не так существенно при малых абсолютных значениях.

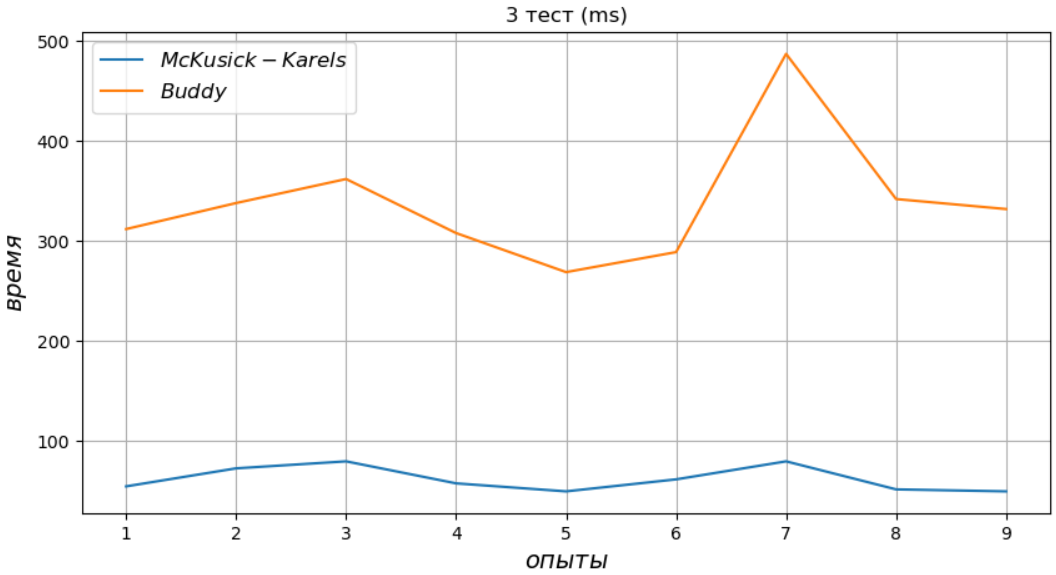
Третий тест проверяет работу алгоритма для большего количества указателей. Из полученных результатов видно, что разница становится гораздо более существенной. Она колеблется от 6 до 7 раз. Из этого можно сделать вывод, что при увеличении количества указателей, алгоритм Мак-Кьюзи-Кэрелса становится гораздо более эффективным, чем алгоритм двойников.

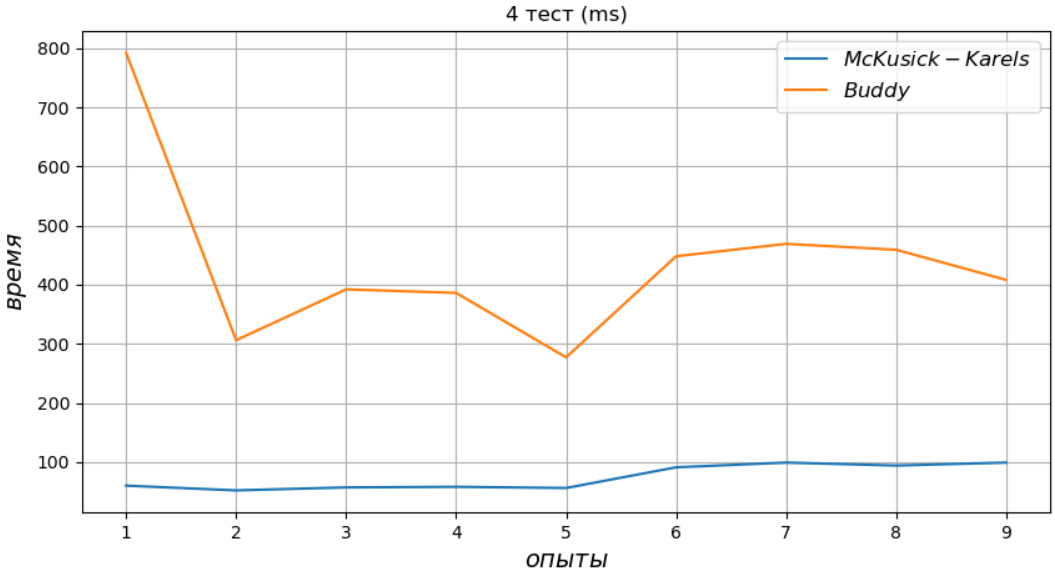
Четвертый тест проверяет, как алгоритмы работают, когда у нас меньше данных записывается в указатели. Из полученных результатов достаточно проблематично сделать выводы, так как данные могут отличаться. Программа может вывести как и то, что программы работают быстрее, так и то, что разница достигает 2 раз, однако обобщая результаты, можно сказать, что время работы увеличивается у обоих алгоритмов, однако из-за того, что алгоритм двойников имеет большее время работы, то на нем это сказывается существеннее.

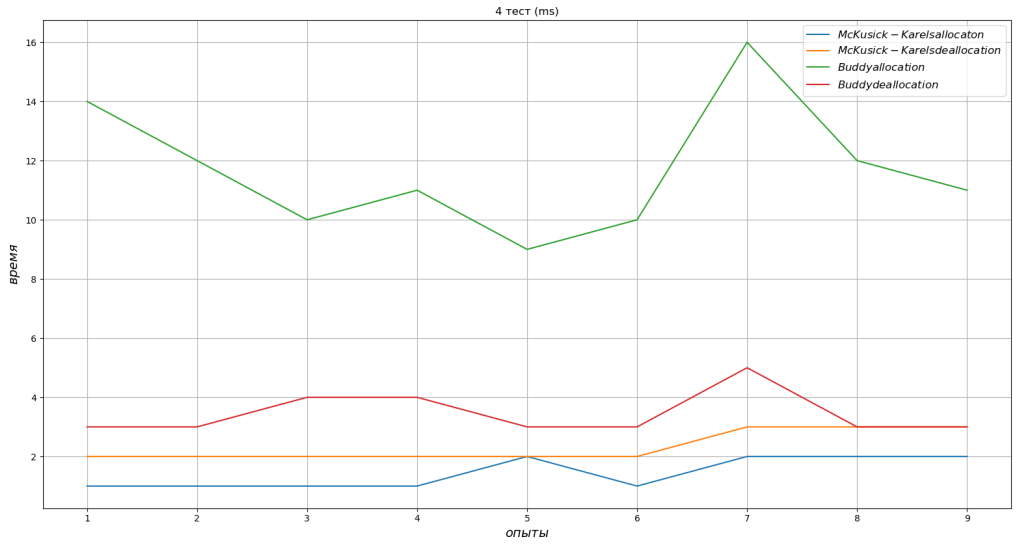
Пятый тест показывает сколько времени каждому алгоритму нужно для аллокации а деаллокации. Из полученных данных можно сказать, что алгоритм Мак-Кьюзи-Кэрелса тратит больше времени на деаллокацию, а алгоритм двойников тратит больше времени а аллокацию.




2 тест (ms)
время
опыты
McKusick – Karels
Buddy


3 тест (ms)
время
опыты
McKusick – Karels
Buddy





Обобщив всё вышесказанное, можно сказать, что хоть и алгоритм двойников быстрее инициализируется, однако сама работа с памятью происходит медленнее, чем Мак-Кьюзи-Кэрелса, причем разница увеличивается тем сильнее, чем больше количество указателей, с которыми мы работаем, поэтому можно сказать, что алгоритм Мак-Кьюзи-Кэрелса предпочтительнее, если нам важна скорость работы, однако в достоинства алгоритма двойников можно сказать о том, что интерфейс работы гораздо более прост.

Выводы

В ходе выполнения курсового проекта я получил навыки работы с аллокаторами, изучил как устроенна память, работают аллокаторы памяти. А также изучил различные алгоритмы аллокации. Также я реализовал 2 аллокатора памяти и сравнил их про быстроте работы. В результате тестов я пришел к выводу о том, что алгоритм Мак-Кьюзи-Кэрелса работает быстрее чем алгоритм двойников, хотя работать с алгоритмом двойников гораздо легче.