Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

**Тема работы**

**Приобретение практических навыков в использовании знаний, полученных в течении курса.**

Студент: Рылов Александр Дмитриевич

Группа: М8О-207Б-21

Вариант: 16

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/Brokiloene/os

**Постановка задачи**

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

Вариант 16: Необходимо сравнить два алгоритма аллокации: алгоритм Мак-Кьюзи-Кэрелса и блоки по 2 в степени n.

**Общие сведения о программе**

Файлы pow\_of\_two.hpp и mckk.hpp содержат интерфейсы классов аллокаторов, файлы pow\_of\_two.сpp и mckk.сpp — их реализацию. Файл main.cpp содержит использование аллокаторов и демонстрацию их работы, сборка осуществляется с помощью утилиты cmake.

**Общий метод и алгоритм решения**

Алгоритм аллокации блоков степени 2 заключается в хранении списков указателей на свободные блоки памяти одного размера, размер блока хранится в самом блоке. Алгоритм Мак-Кьюзи-Кэрелса является улучшенной версией предыдущего, в нём свободные блоки так же хранятся в списках, но размер блока хранится на уровне самого аллокатора в специальном массиве.

**Исходный код**

**main.cpp**

#include <iostream>

#include <chrono>

#include "pow\_of\_two\_allocator.hpp"

#include "mckk.hpp"

using namespace std::chrono;

int main(){

std::vector<int> blocks\_amount = {64, 32, 16, 4, 20, 10, 0};

steady\_clock::time\_point pow2\_init\_start = steady\_clock::now();

pow\_two\_allocator pow\_two\_allocator(blocks\_amount);

steady\_clock::time\_point pow2\_init\_end = steady\_clock::now();

std::cout << "powers of two init time: " <<

std::chrono::duration\_cast<std::chrono::nanoseconds>(pow2\_init\_end - pow2\_init\_start).count() << " nanosec\n";

int pages\_cnt = 10;

std::vector<int> pages\_fragments = {32, 128, 256, 1024, 512, 256, 256, 1024, 16, 256};

steady\_clock::time\_point mckk\_init\_start = steady\_clock::now();

mckk\_allocator mckk\_allocator(pages\_cnt, pages\_fragments);

steady\_clock::time\_point mckk\_init\_end = steady\_clock::now();

std::cout << "mckk init time: " << std::chrono::duration\_cast<std::chrono::nanoseconds>(mckk\_init\_end - mckk\_init\_start).count() << " nanosec\n\n";

std::cout << "test: allocate 10 char[256] + deallocate + allocate 10 char[128] + deallocate:\n";

std::vector<char \*> pointers1(10, 0);

steady\_clock::time\_point pow2\_test\_start = steady\_clock::now();

for (int i = 0; i < 10; ++i) {

pointers1[i] = (char \*)pow\_two\_allocator.allocate(256);

}

for (int i = 0; i < 10; ++i) {

pow\_two\_allocator.deallocate(pointers1[i]);

}

for (int i = 5; i < 10; ++i) {

pointers1[i] = (char \*)pow\_two\_allocator.allocate(128);

}

for (int i = 0; i < 10; ++i) {

pow\_two\_allocator.deallocate(pointers1[i]);

}

steady\_clock::time\_point pow\_two\_test\_end = steady\_clock::now();

std::cerr << "powers of two: " << std::chrono::duration\_cast<std::chrono::microseconds>(pow\_two\_test\_end - pow2\_test\_start).count() << " microsec\n";

std::vector<char \*> pointers2(10, 0);

steady\_clock::time\_point mckk\_test\_start = steady\_clock::now();

for (int i = 0; i < 10; ++i) {

pointers2[i] = (char \*)mckk\_allocator.allocate(256);

}

for (int i = 5; i < 10; ++i) {

mckk\_allocator.deallocate(pointers2[i]);

}

for (int i = 5; i < 10; ++i) {

pointers2[i] = (char \*)mckk\_allocator.allocate(128);

}

for (int i = 0; i < 10; ++i) {

mckk\_allocator.deallocate(pointers2[i]);

}

steady\_clock::time\_point mckk\_test\_end = steady\_clock::now();

std::cerr << "mckk: " << std::chrono::duration\_cast<std::chrono::microseconds>(mckk\_test\_end - mckk\_test\_start).count() << " microsec\n";

}

**mckk.cpp**

#include <iostream>

#include <list>

#include <algorithm>

#include <vector>

#include <map>

#define PAGE\_SIZE 1024

struct Page {

int block\_size;

char \*start;

char \*end;

};

class mckk\_allocator {

private:

std::vector<int> pows\_of\_two = {16,32,64,128,256,512,1024};

std::vector<std::list<char \*>> free\_blocks\_lists;

std::vector<Page> block\_mem\_info;

char \*data;

public:

mckk\_allocator(int pages\_cnt, std::vector<int>& pages\_fragments); // pages\_fragments is a vector with sizes of blocks

// on which page is splitted

void \*allocate(int bytes\_amount);

void deallocate(void \*ptr);

~mckk\_allocator();

};

**pow\_of\_two.cpp**

#include "pow\_of\_two.hpp"

pow\_two\_allocator::pow\_two\_allocator(std::vector<int> &blocks\_amount) {

free\_blocks\_lists = std::vector<std::list<char \*>>(pows\_of\_two.size());

int bytes\_sum = 0;

for (int i = 0; i < blocks\_amount.size(); ++i) {

bytes\_sum += blocks\_amount[i] \* pows\_of\_two[i];

}

data = (char \*) malloc(bytes\_sum);

char \*data\_copy = data;

for (int i = 0; i < blocks\_amount.size(); ++i) {

for (int j = 0; j < blocks\_amount[i]; ++j) {

free\_blocks\_lists[i].push\_back(data\_copy);

\*((int \*)data\_copy) = pows\_of\_two[i];

data\_copy += pows\_of\_two[i];

//std::cout << pows\_of\_two[i] << '\n';

}

}

}

void \*pow\_two\_allocator::allocate(int bytes\_amount) {

if (bytes\_amount == 0) {

return nullptr;

}

bytes\_amount += sizeof(int);

//std::cout << "free\_blocks\_lists.size = " << free\_blocks\_lists.size() << std::endl;

/\*for (auto el : free\_blocks\_lists){

std::cout << el.size() << std::endl;

}\*/

int ind = -1;

for (int i = 0; i < free\_blocks\_lists.size(); ++i) {

if (bytes\_amount <= pows\_of\_two[i] && !free\_blocks\_lists[i].empty()) { // if requested amount of bytes is fit and such block exists

ind = i;

break;

}

}

if (ind == -1) {

std::cout << "There isn't memory\n";

}

char \*memory = free\_blocks\_lists[ind].front();

free\_blocks\_lists[ind].pop\_front();

return (void \*)(memory + sizeof(int));

}

void pow\_two\_allocator::deallocate(void \*ptr) {

char \*char\_ptr = (char \*)ptr;

char\_ptr = char\_ptr - sizeof(int);

int block\_size = \*((int \*)char\_ptr);

int ind = -1;

for(int i = 0; i < pows\_of\_two.size(); ++i) {

if(pows\_of\_two[i] == block\_size) {

ind = i;

}

}

free\_blocks\_lists[ind].push\_back(char\_ptr);

}

pow\_two\_allocator::~pow\_two\_allocator() {

free(data);

}

**pow\_of\_two.hpp**

#include <iostream>

#include <list>

#include <vector>

class pow\_two\_allocator {

private:

std::vector<std::list<char \*>> free\_blocks\_lists;

std::vector<int> pows\_of\_two = {16,32,64,128,256,512,1024};

char \*data;

public:

pow\_two\_allocator(std::vector<int> &blocksAmount);

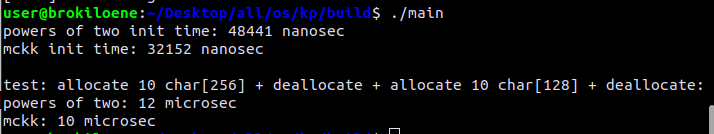
void \*allocate(int bytesAmount);

void deallocate(void \*ptr);

~pow\_two\_allocator();

};

**Демонстрация работы программы**



**Выводы**

В ходе выполнения курсового проекта я закрепил навыки работы с аллокаторами. Из результатов работы программы видно, что инициализация и аллокация алгоритмом Мак-Кьюзи-Кэрелса происходит немного быстрее и поэтому является более предпочтительным алгоритмом аллокации памяти.