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: Полонский Кирилл Андреевич

Группа: М8О-208Б-20

Вариант: 16

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**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/kirillpolonskii/OS/tree/master/os_cp>

**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\_allocator.hpp и mck-k.hpp содержат интерфейсы классов аллокаторов, соответствующих варианту задания, файлы pow\_of\_two\_allocator.сpp и mck-k.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**

pow\_of\_two\_allocator.hpp

#include <iostream>

#include <list>

#include <vector>

class Pow2Allocator {

private:

std::vector<std::list<char\*>> freeBlocksLists;

std::vector<int> powsOf2 = {16,32,64,128,256,512,1024};

char\* data;

public:

Pow2Allocator(std::vector<int>& blocksAmount);

void\* allocate(int bytesAmount);

void deallocate(void \*ptr);

~Pow2Allocator();

};

pow\_of\_two\_allocator.cpp

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

//powOf2 begins with 16

Pow2Allocator::Pow2Allocator(std::vector<int>& blocksAmount){

freeBlocksLists = std::vector<std::list<char\*>>(powsOf2.size());

int bytesSum = 0;

for (int i = 0; i < blocksAmount.size(); ++i){

bytesSum += blocksAmount[i] \* powsOf2[i];

}

data = (char\*)malloc(bytesSum);

char\* dataCopy = data;

for (int i = 0; i < blocksAmount.size(); ++i){

for (int j = 0; j < blocksAmount[i]; ++j){

//std::cout << "In constructor in adding in list\n";

freeBlocksLists[i].push\_back(dataCopy);

\*((int\*)dataCopy) = powsOf2[i];

dataCopy += powsOf2[i];

}

}

}

void\* Pow2Allocator::allocate(int bytesAmount){

if (bytesAmount == 0){

return nullptr;

}

bytesAmount += sizeof(int);

//std::cout << "freeBlocksLists.size = " << freeBlocksLists.size() << std::endl;

*/\*for (auto el : freeBlocksLists){*

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

*}\*/*

int ind = -1;

for (int i = 0; i < freeBlocksLists.size(); ++i){

if (bytesAmount <= powsOf2[i] && !freeBlocksLists[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 = freeBlocksLists[ind].front();

freeBlocksLists[ind].pop\_front();

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

}

void Pow2Allocator::deallocate(void\* ptr){

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

chPtr = chPtr - sizeof(int);

int blockSize = \*((int\*)chPtr);

int ind = -1;

for(int i = 0; i < powsOf2.size(); ++i){

if(powsOf2[i] == blockSize){

ind = i;

}

}

freeBlocksLists[ind].push\_back(chPtr);

}

Pow2Allocator::~Pow2Allocator(){

std::cout << "In destructor**\n**";

free(data);

}

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();

};

mck-k.cpp

#include "mck-k.hpp"

//powOf2 begins with 16

McKKAllocator::McKKAllocator(int pagesAmount, std::vector<int>& pagesFragments){

freeBlocksLists = std::vector<std::list<char\*>>(powsOf2.size());

data = (char\*)malloc(pagesAmount \* PAGE\_SIZE);

char\* curPageStart = data;

char\* curPageEnd = curPageStart + (PAGE\_SIZE - 1);

kMemSize = std::vector<Page>(pagesAmount);

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

kMemSize[i].blockSize = pagesFragments[i];

kMemSize[i].start = curPageStart;

kMemSize[i].end = curPageEnd;

curPageStart += PAGE\_SIZE;

curPageEnd += PAGE\_SIZE;

}

for (int i = 0; i < kMemSize.size(); ++i){

int ind = -1;

for(int j = 0; j < powsOf2.size(); ++j){

if(kMemSize[i].blockSize == powsOf2[j]){

ind = j;

break;

}

}

char\* curBlockStart = kMemSize[i].start;

for(int j = 0; j < PAGE\_SIZE / kMemSize[i].blockSize; ++j){

//std::cout << "In constructor in adding in list\n";

freeBlocksLists[ind].push\_back(curBlockStart);

curBlockStart += kMemSize[i].blockSize;

}

}

}

void\* McKKAllocator::allocate(int bytesAmount){

if (bytesAmount == 0){

return nullptr;

}

//std::cout << "1freeBlocksLists.size = " << freeBlocksLists.size() << std::endl;

*/\*for (auto el : freeBlocksLists){*

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

*}\*/*

int ind = -1;

for (int i = 0; i < freeBlocksLists.size(); ++i){

if (bytesAmount <= powsOf2[i] && !freeBlocksLists[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 = freeBlocksLists[ind].front();

freeBlocksLists[ind].pop\_front();

return (void\*)memory;

}

void McKKAllocator::deallocate(void\* ptr){

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

int indPage = -1;

for(int i = 0; i < kMemSize.size(); ++i){

if(kMemSize[i].start <= chPtr && chPtr <= kMemSize[i].end){

indPage = i;

break;

}

}

int indBlock = -1;

for(int j = 0; j < powsOf2.size(); ++j){

if(kMemSize[indPage].blockSize == powsOf2[j]){

indBlock = j;

break;

}

}

freeBlocksLists[indBlock].push\_back(chPtr);

}

McKKAllocator::~McKKAllocator(){

std::cout << "In destructor1**\n**";

free(data);

}

main.cpp

#include <iostream>

#include <chrono>

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

#include "mck-k.hpp"

int main(){

using namespace std::chrono;

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

steady\_clock::time\_point pow2AllocatorInitStart = steady\_clock::now();

Pow2Allocator pow2Allocator(blocksAmount);

steady\_clock::time\_point pow2AllocatorInitEnd = steady\_clock::now();

std::cout << "Powers-of-2 allocator initialization: " <<

std::chrono::duration\_cast<std::chrono::nanoseconds>(pow2AllocatorInitEnd - pow2AllocatorInitStart).count() << " ns" << std::endl;

std::cout << 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 allocator initialization: " <<

std::chrono::duration\_cast<std::chrono::nanoseconds>(mcKKAllocatorInitEnd - mcKKAllocatorInitStart).count() << " ns" << std::endl;

std::cout << std::endl;

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

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

steady\_clock::time\_point pow2TestStart = steady\_clock::now();

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

pointers1[i] = (char\*)pow2Allocator.allocate(256);

}

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

pow2Allocator.deallocate(pointers1[i]);

}

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

pointers1[i] = (char\*)pow2Allocator.allocate(128);

}

steady\_clock::time\_point pow2TestEnd = steady\_clock::now();

std::cerr << "Powers-of-2 allocator test:" <<

std::chrono::duration\_cast<std::chrono::microseconds>(pow2TestEnd - pow2TestStart).count() << " microseconds" << std::endl;

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

pow2Allocator.deallocate(pointers1[i]);

}

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 allocator test:" <<

std::chrono::duration\_cast<std::chrono::microseconds>(mcKKTest1End - mcKKTest1Start).count() << " microseconds" << std::endl;

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

mcKKAllocator.deallocate(pointers2[i]);

}

}

**Демонстрация работы программы**

| Powers-of-2 allocator initialization: 340505 ns  McKusick-Karels allocator initialization: 353618 ns  Test: allocate 10 char[256], deallocate 5 of them, allocate 5 char[128]:  Powers-of-2 allocator test:49 microseconds  McKusick-Karels allocator test:50 microseconds  In destructor1  In destructor |
| --- |

**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. По фактору использования алгоритм блоков степени 2 выигрывает лишь в случае, когда запрашиваемые блоки будут иметь размер 2n - 4: тогда память будет распределяться наиболее оптимально.