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

Курсовой проект по курсу
«Операционные системы»

Студент: Останина Анна Андреевна
Группа: М8О-208Б-22
Вариант: 16
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

Репозиторий

https://github.com/Imariiii/os_labs

Цель работы

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

Задание

Необходимо сравнить два алгоритма аллокации: выделение памяти по степени 2 и алгоритм Мак-Кьюзи-Кэрелса

Описание работы программы

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.

Выделение памяти по степени 2:

- Вся память делится на списки свободных элементов равного размера.
- Если элемент свободен, то он хранит ссылку на следующий свободный элемент.
- Если элемент занят, то хранится ссылка на голову списка, откуда он был взят.
- При освобождении нужно просто добавить элемент к голове указанного списка.
- При выделении выбирается список K = [log2(N)].

Алгоритм Мак-Кьюзи-Кэрелса:

Алгоритм подразумевает, что память разбита на набор последовательных страниц, и все буферы, относящиеся к одной странице, должны иметь

одинаковый размер (являющийся некоторой степенью числа 2). Каждая страница может находиться в одном из трёх перечисленных состояний.

- Быть свободной.
- Быть разбитой на буферы определённого размера.
- Являться частью буфера, объединяющего сразу несколько страниц.

Исходный код

mck_allocator.h

```h
#ifndef POW_ALLOCATOR_H
#define MCK_ALLOCATOR_H
#include <stddef.h>
#include <stdbool.h>

struct Page {
    struct Page* next;
    bool isLarge;
    size_t blockSize;
};

struct MCKAllocator {
    struct Page* freePagesList;
    size_t pageSize;
};

void MCKInit(struct MCKAllocator* a);
void MCKDestroy(struct MCKAllocator* a);
void *MCKAlloc(struct MCKAllocator* a, size_t newblockSize);
void MCKFree(struct MCKAllocator* a, void* block);
#endif  // MCK_ALLOCATOR_H
```

mck_allocator.c

```c
#include "mck_allocator.h"

#include <sys/mman.h>
#include <unistd.h>

void MCKInit(struct MCKAllocator* a) {
    a->freePagesList = NULL;
    a->pageSize = getpagesize();
}

void MCKDestroy(struct MCKAllocator* a) {
    struct Page* curPage = a->freePagesList;

    while (curPage) {
        struct Page* toDelete = curPage;
        curPage = curPage->next;
        munmap(toDelete, a->pageSize);
        toDelete = NULL;
    }
    a->freePagesList = NULL;
}
```

```
void *MCKAlloc(struct MCKAllocator* a, size_t newblockSize) {

    size_t roundedBlockSize = 1;
    while (roundedBlockSize < newblockSize) {
        roundedBlockSize *= 2;
    }

    struct Page* curPage = a->freePagesList;

    while (curPage) {
        if (!curPage->isLarge &&
            curPage->blockSize == roundedBlockSize) {
            void* block = (void*)(curPage);
            a->freePagesList = curPage->next;

            return block;
        }

        curPage = curPage->next;
    }


    struct Page* newPage =
        (struct Page*)(mmap(NULL, a->pageSize, PROT_READ | PROT_WRITE,
                                    MAP_PRIVATE | MAP_ANONYMOUS, -1, 0));

    if (newPage == MAP_FAILED) {
        return NULL;
    }
    newPage->isLarge = false;
    newPage->blockSize = roundedBlockSize;
    newPage->next = NULL;

    size_t numBlocks = a->pageSize / roundedBlockSize;
    for (size_t i = 0; i != numBlocks; ++i) {
        struct Page* blockPage = (struct Page*)(
            (char*)newPage + i * roundedBlockSize);
        blockPage->isLarge = false;
        blockPage->blockSize = roundedBlockSize;
        blockPage->next = a->freePagesList;
        a->freePagesList = blockPage;
    }

    void* block = (void*)(newPage);
    a->freePagesList = newPage->next;

    return block;
}

void MCKFree(struct MCKAllocator* a, void* block) {
    if (block == NULL) return;

    struct Page* page = (struct Page*)(block);
    page->next = a->freePagesList;
    a->freePagesList = page;
}

pow_allocator.h
#ifndef POW_ALLOCATOR_H
#define POW_ALLOCATOR_H
#include <stddef.h>
```

```
#define INIT_POW 5
#define NUM_OF_POWS 13


void PowInit();
void *PowAlloc(size_t size);
void PowFree(void *ptr);

#endif  // POW_ALLOCATOR_H
```

pow_allocator.c

```
#include "pow_allocator.h"

#include <math.h>
#include <stdio.h>

struct Elem {
    void *ptr;
    char data[];
};

static void *HEADS[NUM_OF_POWS];
static char POOL[100000000];

void PowInit() {
    char *data = POOL;
    for (int i = 0; i < NUM_OF_POWS; ++i) {
        HEADS[i] = data;
        int pow = INIT_POW;
        size_t sizeofBlock = (1 << pow) + sizeof(void *);
        for (int j = 0; j < 100000; ++j) {
            ((struct Elem *)data)->ptr = data + sizeofBlock;
            data += sizeofBlock;
        }
        ((struct Elem *)data)->ptr = NULL;
        pow++;
        data += sizeofBlock;
    }
}

void *PowAlloc(size_t size) {
    int k;
    if (size < (1 << NUM_OF_POWS) + 1) {
        k = 0;
    } else {
        k = ceil(log2(size)) - INIT_POW;
    }
    struct Elem *temp = HEADS[k];
    HEADS[k] = temp->ptr;
    temp->ptr=HEADS+k;
    return temp->data;
}

void PowFree(void *ptr) {
    struct Elem *temp = (struct Elem *)((char*)ptr-sizeof(void*));
    void **headPtr = temp->ptr;
    temp->ptr = *headPtr;
    *headPtr = temp;
}
```

main.cpp

```
#include <unistd.h>
```

```
#include <chrono>
#include <cstdlib>
#include <iostream>
#include <vector>

extern "C" {
#include "mck_allocator.h"
#include "pow_allocator.h"
}
size_t page_size = sysconf(_SC_PAGESIZE);

static void benchmark(struct MCKAllocator* MCKAlloctor, std::size_t size,
                      std::size_t n) {
    PowInit();
    MCKInit(MCKAlloctor);
    std::vector<void*> list_blocks;
    std::vector<void*> MKC_blocks;

    std::cout << "Comparing PowAllocator and MCKAllocator" << std::endl;

    std::cout << "Block allocation rate of " << n << " chunks of " << size
              << " bytes" << std::endl;
    auto start_time = std::chrono::steady_clock::now();
    for (size_t i = 0; i != n; ++i) {
        void* block = PowAlloc(size);
        list_blocks.push_back(block);
    }
    auto end_time = std::chrono::steady_clock::now();
    std::cout << "Time of alloc PowAllocator: "
              << 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 != n; ++i) {
        void* block = MCKAlloc(MCKAlloctor, size);
        MKC_blocks.push_back(block);
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of alloc MCKAllocator: "
              << 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) {
        PowFree(list_blocks[i]);
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of free PowAllocator: "
              << 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) {
        MCKFree(MCKAlloctor, MKC_blocks[i]);
    }
    end_time = std::chrono::steady_clock::now();
```

```
    std::cout << "Time of free MCKAllocator: "
              << std::chrono::duration_cast<std::chrono::milliseconds>(
                     end_time - start_time)
                     .count()
              << " milliseconds" << std::endl;
}

int main() {
    MCKAllocator a;

    for (size_t i = 0; i < 10; ++i)
    benchmark(&a, 100*(i+1), 70000);

    std::cout << "-----------------------";
    for (size_t i = 0; i < 5; ++i)
    benchmark(&a, 1000, 10000*(i+1));


}
```

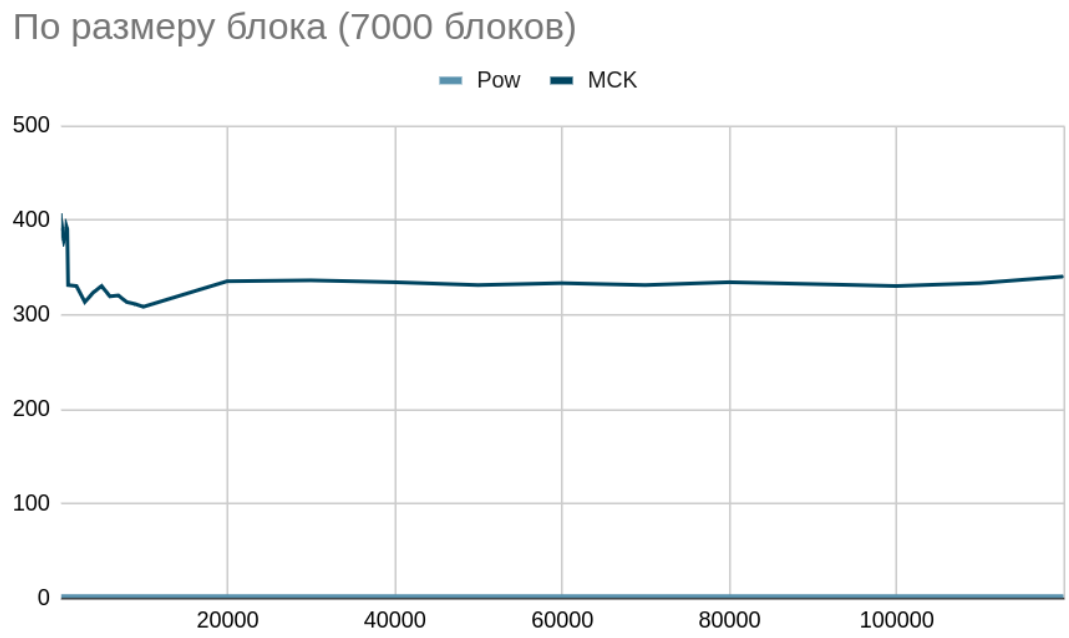
Консоль

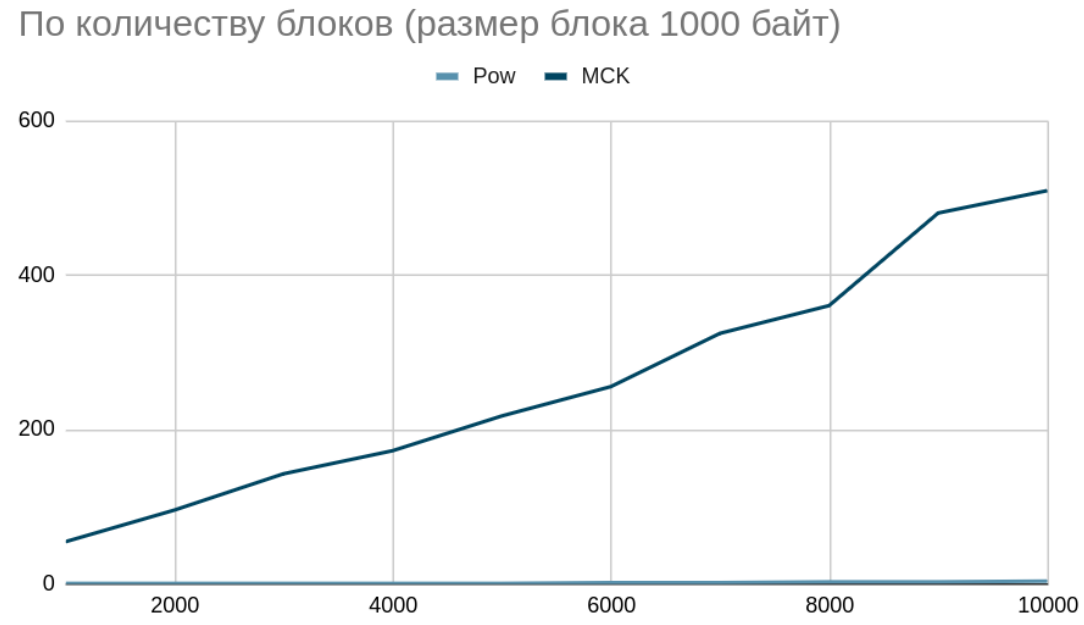
```
anna@anna-virtual-machine:~/labs_3sem/os_labs/build/cp$ ./cp
Comparing PowAllocator and MCKAllocator
Block allocation rate of 70000 chunks of 100 bytes
Time of alloc PowAllocator: 6 milliseconds
Time of alloc MCKAllocator: 815 milliseconds
Block free rate
Time of free PowAllocator: 0 milliseconds
Time of free MCKAllocator: 5 milliseconds
Comparing PowAllocator and MCKAllocator
Block allocation rate of 70000 chunks of 200 bytes
Time of alloc PowAllocator: 5 milliseconds
Time of alloc MCKAllocator: 1343 milliseconds
Block free rate
Time of free PowAllocator: 1 milliseconds
Time of free MCKAllocator: 3 milliseconds
Comparing PowAllocator and MCKAllocator
Block allocation rate of 70000 chunks of 300 bytes
Time of alloc PowAllocator: 3 milliseconds
Time of alloc MCKAllocator: 1000 milliseconds
Block free rate
Time of free PowAllocator: 1 milliseconds
Time of free MCKAllocator: 3 milliseconds
Comparing PowAllocator and MCKAllocator
Block allocation rate of 70000 chunks of 400 bytes
Time of alloc PowAllocator: 4 milliseconds
Time of alloc MCKAllocator: 955 milliseconds
Block free rate
Time of free PowAllocator: 0 milliseconds
Time of free MCKAllocator: 2 milliseconds
Comparing PowAllocator and MCKAllocator
Block allocation rate of 70000 chunks of 500 bytes
Time of alloc PowAllocator: 4 milliseconds
Time of alloc MCKAllocator: 948 milliseconds
Block free rate
Time of free PowAllocator: 1 milliseconds
Time of free MCKAllocator: 4 milliseconds
Comparing PowAllocator and MCKAllocator
Block allocation rate of 70000 chunks of 600 bytes
Time of alloc PowAllocator: 7 milliseconds
Time of alloc MCKAllocator: 949 milliseconds
Block free rate
```

```
Time of free PowAllocator: 1 milliseconds
Time of free MCKAllocator: 3 milliseconds
Comparing PowAllocator and MCKAllocator
Block allocation rate of 70000 chunks of 700 bytes
Time of alloc PowAllocator: 3 milliseconds
Time of alloc MCKAllocator: 1138 milliseconds
Block free rate
Time of free PowAllocator: 0 milliseconds
Time of free MCKAllocator: 3 milliseconds
Comparing PowAllocator and MCKAllocator
Block allocation rate of 70000 chunks of 800 bytes
Time of alloc PowAllocator: 13 milliseconds
Time of alloc MCKAllocator: 1390 milliseconds
Block free rate
Time of free PowAllocator: 1 milliseconds
Time of free MCKAllocator: 4 milliseconds
Comparing PowAllocator and MCKAllocator
Block allocation rate of 70000 chunks of 900 bytes
Time of alloc PowAllocator: 5 milliseconds
Time of alloc MCKAllocator: 1790 milliseconds
Block free rate
Time of free PowAllocator: 1 milliseconds
Time of free MCKAllocator: 15 milliseconds
Comparing PowAllocator and MCKAllocator
Block allocation rate of 70000 chunks of 1000 bytes
Time of alloc PowAllocator: 11 milliseconds
Time of alloc MCKAllocator: 5096 milliseconds
Block free rate
Time of free PowAllocator: 4 milliseconds
Time of free MCKAllocator: 27 milliseconds
----------------------Comparing PowAllocator and MCKAllocator
Block allocation rate of 10000 chunks of 1000 bytes
Time of alloc PowAllocator: 2 milliseconds
Time of alloc MCKAllocator: 575 milliseconds
Block free rate
Time of free PowAllocator: 0 milliseconds
Time of free MCKAllocator: 0 milliseconds
Comparing PowAllocator and MCKAllocator
Block allocation rate of 20000 chunks of 1000 bytes
Time of alloc PowAllocator: 0 milliseconds
Time of alloc MCKAllocator: 1373 milliseconds
Block free rate
Time of free PowAllocator: 7 milliseconds
Time of free MCKAllocator: 0 milliseconds
Comparing PowAllocator and MCKAllocator
Block allocation rate of 30000 chunks of 1000 bytes
Time of alloc PowAllocator: 2 milliseconds
Time of alloc MCKAllocator: 2263 milliseconds
Block free rate
Time of free PowAllocator: 0 milliseconds
Time of free MCKAllocator: 6 milliseconds
Comparing PowAllocator and MCKAllocator
Block allocation rate of 40000 chunks of 1000 bytes
Time of alloc PowAllocator: 20 milliseconds
Time of alloc MCKAllocator: 2359 milliseconds
Block free rate
Time of free PowAllocator: 1 milliseconds
Time of free MCKAllocator: 4 milliseconds
Comparing PowAllocator and MCKAllocator
Block allocation rate of 50000 chunks of 1000 bytes
Time of alloc PowAllocator: 2 milliseconds
Time of alloc MCKAllocator: 3195 milliseconds
Block free rate
```

```
Time of free PowAllocator: 443 milliseconds
Time of free MCKAllocator: 13 milliseconds
```

Графики





Выводы

В ходе данного курсового проекта были проведены исследования двух аллокаторов памяти:

Выделение памяти по степени 2 и Алгоритм Мак-Кьюзи-Кэрелса.

Сравнивая эти два способа аллокации памяти, основываясь на времени их работы и на самом принципе работы, можно сделать вывод, что метод выделения памяти по степени 2 быстрее, чем алгоритм Мак-Кьюзи-Кэрелса, так как второй алгоритм использует страничный аллокатор, а первый выделяет пул заранее. С другой стороны, второй алгоритм не имеет ограничений по памяти (блоки можно выделять нужного размера, и они формируются по требованиям системы) и выделяет новые страницы только при необходимости. Оба алгоритма не требуют указывать размер освобождаемого блока.