Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа № 4 по курсу

«Операционные системы»

Группа: М8О-214Б-23

Студент: Шестаков К. Р.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 30.12.24

Москва, 2024

Постановка задачи

Вариант 5.

Исследовать два аллокатора памяти: необходимо реализовать два алгоритма аллокации памяти и сравнить их по следующим характеристикам:
– Фактор использования
– Скорость выделения блоков
– Скорость освобождения блоков
– Простота использования аллокатора
Требуется создать две динамические библиотеки, реализующие два аллокатора, соответственно. Библиотеки загружаются в память с помощью интерфейса ОС (dlopen / LoadLibrary) для работы с динамическими библиотеками. Выбор библиотеки, реализующей аллокатор, осуществляется чтением первого аргумента при запуске программы (argv[1]). Этот аргумент должен содержать путь до динамической библиотеки (относительный или абсолютный).
Если аргумент не передан или по переданному пути библиотеки не оказалось, то указатели на функции, реализующие API аллокатора ниже, должны быть присвоены функциям, которые оборачивают системный аллокатор ОС (mmap / VirtualAlloc) в этот API. Эти аварийные оберточные функции должны быть реализованы внутри программы, которая загружает динамические библиотеки (см. пример на GitHub Gist).
Каждый аллокатор памяти должен иметь функции аналогичные стандартным функциям malloc и free (realloc, опционально). Перед работой каждый аллокатор инициализируется свободными страницами памяти, выделенными стандартными средствами ядра (mmap / VirtualAlloc). 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.

Алгоритм Мак-Кьюзика-Кэрелса и алгоритм двойников

Общий метод и алгоритм решения

Использованные системные вызовы Windows:

VirtualAlloc: Выделение блока виртуальной памяти заданного размера. Используется для получения памяти под аллокаторы и их нужды.

VirtualFree: Освобождение блока памяти, выделенного через VirtualAlloc.

LoadLibraryA: Загрузка динамической библиотеки (DLL) в память процесса. Применяется для загрузки библиотек с реализациями аллокаторов.

GetProcAddress: Получение адреса экспортируемой функции из загруженной DLL. Используется для доступа к функциям API аллокаторов.

FreeLibrary: Выгрузка динамической библиотеки из памяти процесса.

Работа программы main.c:

Программа тестирует два алгоритма аллокации памяти, реализованных в отдельных DLL.

Выбор аллокатора: При запуске проверяет аргумент командной строки. Если он есть и указывает на существующую DLL, загружает её (LoadLibraryA). В противном случае использует обертки вокруг VirtualAlloc/VirtualFree.

Получение функций: Из загруженной DLL (GetProcAddress) или из оберток получает указатели на функции API аллокатора (allocator_create, allocator_destroy, allocator_alloc, allocator_free).

Инициализация: Выделяет память (VirtualAlloc) и инициализирует выбранный аллокатор (allocator_create).

Бенчмарк: Многократно выделяет (allocator_alloc) и освобождает (allocator_free) блоки памяти случайных размеров, измеряя время выполнения этих операций.

Деинициализация: Деинициализирует аллокатор (allocator_destroy) и выгружает DLL (FreeLibrary), если она загружалась.

Вывод результатов: Выводит информацию о времени выделения и освобождения памяти, а также факторе использования памяти.

Подробное описание каждого из исследуемых алгоритмов

1. Алгоритм Buddy System: (алгоритм двойников)

Алгоритм Buddy System является методом управления памятью, основанным на выделении блоков памяти размером, равным степени двойки. Основная идея заключается в рекурсивном разделении больших блоков свободной памяти на два равных блока-«близнеца» до тех пор, пока не будет найден блок подходящего размера для запроса.

Выделение памяти: При поступлении запроса на выделение памяти ищется свободный блок, размер которого является наименьшей степенью двойки, большей или равной запрошенному размеру. Если такого блока нет, ищется блок большего размера, который затем рекурсивно делится на пары «близнецов», пока не будет получен блок нужного размера. Один из «близнецов» выделяется, а другой остается свободным.

Освобождение памяти: При освобождении блока памяти проверяется, является ли его «близнец» также свободным. Если да, то оба блока объединяются в блок большего размера, и этот процесс повторяется рекурсивно до тех пор, пока «близнец» не окажется занятым или не будет достигнут максимальный размер блока.

Преимущества: Простота реализации, быстрое выделение и освобождение (операции деления и объединения сводятся к простым арифметическим и битовым операциям).

Недостатки: Внутренняя фрагментация (выделяется блок размером, кратным степени двойки, что может привести к неиспользуемому пространству внутри блока), потенциальные проблемы с внешней фрагментацией в определенных сценариях.

2. Алгоритм Мак-Кьюзика-Кэрелса:

Алгоритм Мак-Кьюзика-Кэрелса (также известный как менеджер свободных списков) является более гибким подходом к управлению памятью. Он поддерживает список свободных блоков памяти переменного размера.

Выделение памяти: При поступлении запроса на выделение памяти происходит поиск в списке свободных блоков. Наиболее распространенные стратегии поиска:

First-fit: Выделяется первый найденный блок, размер которого достаточен.

Best-fit: Выделяется блок, размер которого наиболее близок к запрошенному размеру.

Worst-fit: Выделяется самый большой доступный блок.

Если найденный блок значительно больше запрошенного размера, он может быть разделен на два блока: один выделяется, а оставшийся блок возвращается в список свободных.

Освобождение памяти: При освобождении блока он возвращается в список свободных блоков. Важным аспектом является слияние освобожденного блока с соседними свободными блоками для предотвращения фрагментации. Проверяются соседние блоки слева и справа, и если они свободны, происходит слияние в один большой свободный блок.

Преимущества: Меньшая внутренняя фрагментация (блоки выделяются практически точно в соответствии с запрошенным размером), эффективное использование памяти.

Недостатки: Более сложная реализация по сравнению с Buddy System, потенциально более медленное выделение и освобождение (необходимость поиска в списке свободных блоков и выполнение операций слияния).

Процесс тестирования

Для тестирования реализованных аллокаторов памяти был разработан следующий процесс:

1. Динамическая загрузка библиотек: Программа `main.c` принимает в качестве аргумента путь к динамической библиотеке (`.dll`). Используются системные вызовы `LoadLibraryA` для

загрузки библиотеки и `GetProcAddress` для получения указателей на функции API аллокатора (`allocator_create`, `allocator_destroy`, `allocator_alloc`, `allocator_free`).

2. Инициализация аллокатора: После загрузки библиотеки выделяется статический буфер памяти (`MEMORY_SIZE`) с помощью `VirtualAlloc`. Затем вызывается функция `allocator_create` из загруженной библиотеки для инициализации аллокатора в выделенном буфере.

3. Серия выделений и освобождений: Выполняется заданное количество (`ALLOCATIONS`) итераций. На каждой итерации генерируется случайный размер блока (`rand() % MAX_ALLOC_SIZE + 1`). Затем вызывается функция `allocator_alloc` для выделения блока памяти этого размера. Выделенные блоки сохраняются в массиве `allocations`.

4. Измерение времени выделения: Измеряется общее время, затраченное на все операции выделения, с использованием высокоточного таймера (`QueryPerformanceCounter`).

5. Освобождение выделенных блоков: После всех выделений выполняется последовательное освобождение всех выделенных ранее блоков с помощью функции `allocator_free`.

6. Измерение времени освобождения: Аналогично времени выделения, измеряется общее время, затраченное на операции освобождения.

7. Расчет фактора использования: После всех операций выделения рассчитывается общий объем успешно выделенной памяти и делится на общий размер доступной памяти (`MEMORY_SIZE`) для определения фактора использования.

8. Деинициализация и выгрузка: После тестирования вызываются функции `allocator_destroy` для деинициализации аллокатора и `FreeLibrary` для выгрузки динамической библиотеки.

9. Тестирование системного аллокатора: Если при запуске не был указан путь к динамической библиотеке или загрузка библиотеки не удалась, программа использует обертки вокруг системных функций `VirtualAlloc` и `VirtualFree` для имитации работы аллокатора. Это позволяет сравнить производительность пользовательских аллокаторов с системным.

Обоснование подхода тестирования

Выбранный подход к тестированию основан на следующих принципах:

Реалистичная нагрузка: Моделируется ситуация, когда приложение выполняет многократные выделения и освобождения блоков памяти различных размеров, что типично для многих программ. Использование случайных размеров выделяемых блоков приближает нагрузку к реальным условиям.

Изоляция аллокатора: Использование динамических библиотек позволяет изолировать реализацию каждого аллокатора и избежать влияния других частей программы на результаты тестирования.

Измерение ключевых характеристик: Основное внимание уделяется измерению времени выделения и освобождения памяти, поскольку это критически важные параметры производительности аллокаторов. Фактор использования также является важной характеристикой, показывающей эффективность использования доступной памяти.

Сравнение с системным аллокатором: Предоставление возможности использовать системный аллокатор в качестве "резервного" варианта позволяет получить базовые показатели производительности и сравнить с ними пользовательские реализации. Это помогает оценить, насколько разработанные аллокаторы эффективнее (или менее эффективны) стандартных средств.

Минимизация накладных расходов измерения: Измерение общего времени для большого количества операций (ALLOCATIONS) позволяет усреднить случайные колебания и уменьшить влияние накладных расходов, связанных с вызовами функций измерения времени.

Результаты тестирования

Результаты тестирования представлены ниже:

Аллокатор	Время выделения (секунды)	Время освобождения (секунды)	Фактор использования
Мак-Кьюзика-Кэрелса	0.000223	0.000008	0.994447
Buddy System	0.000525	0.000037	0.747251
Системный (VirtualAlloc)	0.002842	0.000008	N/A (нет выделений)

Анализ результатов:

Фактор использования: Алгоритм Мак-Кьюзика-Кэрелса демонстрирует значительно более высокий фактор использования памяти по сравнению с Buddy System. Это связано с тем, что он позволяет выделять блоки более точно под размер запроса, минимизируя внутреннюю фрагментацию. Buddy System, в свою очередь, выделяет блоки размером, кратным степени двойки, что приводит к большему количеству неиспользуемого пространства внутри выделенных блоков. Системный аллокатор в данном тесте не смог выделить память, вероятно из-за подхода к "резервному" выполнению и отсутствия логики управления выделенной памятью в представленном коде обертки.

Скорость выделения: Алгоритм Мак-Кьюзика-Кэрелса показал более высокую скорость выделения памяти по сравнению с Buddy System. Это может быть связано с более простой логикой поиска и выделения блоков в данном конкретном наборе тестов. Системный аллокатор показал значительно более низкую скорость выделения.

Скорость освобождения: Время освобождения у алгоритма Мак-Кьюзика-Кэрелса и системного аллокатора примерно одинаково и значительно ниже, чем у Buddy System. Это может указывать на более эффективную реализацию процесса освобождения в этих алгоритмах в данном тесте.

Код программы

allocator.h

```c
#ifndef ALLOCATOR_H
#define ALLOCATOR_H

#include <stddef.h>

typedef struct Allocator Allocator;

Allocator *allocator_create(void *const memory, const size_t size);
void allocator_destroy(Allocator *const allocator);
void *allocator_alloc(Allocator *const allocator, const size_t size);
void allocator_free(Allocator *const allocator, void *const memory);

#endif
```

buddy.c

```c
#include "allocator.h"
#include <stddef.h>
#include <math.h>
#include <windows.h>

#define MIN_ORDER 4
#define ALIGNMENT 8

typedef struct FreeBlock
{
    struct FreeBlock *next;
} FreeBlock;

struct Allocator
{
    void *memory;
    size_t size;
    int max_order;
    FreeBlock **free_lists;
};

static void *sys_alloc(size_t size)
{
    return VirtualAlloc(NULL, size, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
}

static void sys_free(void *ptr, size_t size)
{
    VirtualFree(ptr, 0, MEM_RELEASE);
}

static size_t align(size_t size)
{
    return (size + ALIGNMENT - 1) & ~(ALIGNMENT - 1);
}
```

```
static int get_order(size_t size)
{
    int order = MIN_ORDER;
    size_t block_size = 1 << MIN_ORDER;
    while (block_size < size)
    {
        block_size <<= 1;
        order++;
    }
    return order;
}

static void split_block(Allocator *allocator, int order)
{
    int higher_order = order + 1;
    FreeBlock *block = allocator->free_lists[higher_order];
    allocator->free_lists[higher_order] = block->next;

    size_t block_size = 1 << order;
    FreeBlock *buddy = (FreeBlock *)((char *)block + block_size);

    block->next = buddy;
    buddy->next = allocator->free_lists[order];
    allocator->free_lists[order] = block;
}

Allocator *allocator_create(void *const memory, const size_t size)
{
    if (memory == NULL || size == 0)
    {
        return NULL;
    }

    Allocator *allocator = (Allocator *)memory;
    allocator->memory = (char *)memory + sizeof(Allocator);
    allocator->size = size - sizeof(Allocator);
    allocator->max_order = get_order(allocator->size);

    allocator->free_lists = (FreeBlock **)sys_alloc(sizeof(FreeBlock *) * (allocator-
>max_order + 1));
    if (allocator->free_lists == NULL)
    {
        return NULL;
    }

    for (int i = 0; i <= allocator->max_order; i++)
    {
        allocator->free_lists[i] = NULL;
    }

    allocator->free_lists[allocator->max_order] = (FreeBlock *)allocator->memory;
    allocator->free_lists[allocator->max_order]->next = NULL;
```

```c
    return allocator;
}

void allocator_destroy(Allocator *const allocator)
{
    if (allocator != NULL && allocator->free_lists != NULL)
    {
        sys_free(allocator->free_lists, sizeof(FreeBlock *) * (allocator->max_order + 1));
    }
}

void *allocator_alloc(Allocator *const allocator, const size_t size)
{
    if (allocator == NULL || size == 0)
    {
        return NULL;
    }

    size_t alloc_size = align(size + sizeof(size_t));
    int order = get_order(alloc_size);

    if (order > allocator->max_order)
    {
        return NULL;
    }

    if (allocator->free_lists[order] == NULL)
    {
        int i = order + 1;
        while (i <= allocator->max_order && allocator->free_lists[i] == NULL)
        {
            i++;
        }

        if (i > allocator->max_order)
        {
            return NULL;
        }

        while (i > order)
        {
            split_block(allocator, i - 1);
            i--;
        }
    }

    FreeBlock *block = allocator->free_lists[order];
    allocator->free_lists[order] = block->next;

    size_t *block_size = (size_t *)block;
    *block_size = 1 << order;

    return (char *)block + sizeof(size_t);
}
```

```
void *get_buddy(Allocator *allocator, void *block, int order)
{
    size_t block_size = 1 << order;
    size_t block_offset = (char *)block - (char *)allocator->memory;
    size_t buddy_offset = block_offset ^ block_size;
    return (char *)allocator->memory + buddy_offset;
}

void allocator_free(Allocator *const allocator, void *const memory)
{
    if (allocator == NULL || memory == NULL)
    {
        return;
    }

    size_t *block_size_ptr = (size_t *)((char *)memory - sizeof(size_t));
    size_t block_size = *block_size_ptr;
    int order = get_order(block_size);
    FreeBlock *block = (FreeBlock *)((char *)memory - sizeof(size_t));

    while (order < allocator->max_order)
    {
        FreeBlock *buddy = (FreeBlock *)get_buddy(allocator, block, order);

        FreeBlock *current = allocator->free_lists[order];
        FreeBlock *prev = NULL;
        while (current != NULL && current != buddy)
        {
            prev = current;
            current = current->next;
        }

        if (current == NULL)
        {
            break;
        }

        if (prev == NULL)
        {
            allocator->free_lists[order] = current->next;
        }
        else
        {
            prev->next = current->next;
        }

        if (buddy < block)
        {
            block = buddy;
        }

        order++;
        block_size <<= 1;
```

```
    }

    block->next = allocator->free_lists[order];
    allocator->free_lists[order] = block;
}
```

mckusick_karels.c

```
#include "allocator.h"
#include <stddef.h>
#include <windows.h>

#define MIN_ALLOCATION_SIZE 16
#define ALIGNMENT 8

typedef struct FreeBlock
{
    size_t size;
    struct FreeBlock *next;
} FreeBlock;

struct Allocator
{
    void *memory;
    size_t size;
    FreeBlock *free_list;
};

static void *sys_alloc(size_t size)
{
    return VirtualAlloc(NULL, size, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
}

static void sys_free(void *ptr, size_t size)
{
    VirtualFree(ptr, 0, MEM_RELEASE);
}

static size_t align(size_t size)
{
    return (size + ALIGNMENT - 1) & ~(ALIGNMENT - 1);
}

Allocator *allocator_create(void *const memory, const size_t size)
{
    if (memory == NULL || size == 0)
    {
        return NULL;
    }

    Allocator *allocator = (Allocator *)memory;
    allocator->memory = (char *)memory + sizeof(Allocator);
    allocator->size = size - sizeof(Allocator);
    allocator->free_list = (FreeBlock *)allocator->memory;
```

```
    allocator->free_list->size = allocator->size;
    allocator->free_list->next = NULL;

    return allocator;
}

void allocator_destroy(Allocator *const allocator)
{
}

void *allocator_alloc(Allocator *const allocator, const size_t size)
{
    if (allocator == NULL || size == 0)
    {
        return NULL;
    }

    size_t alloc_size = align(size + sizeof(size_t));

    FreeBlock *current = allocator->free_list;
    FreeBlock *prev = NULL;

    while (current != NULL)
    {
        if (current->size >= alloc_size)
        {

            if (current->size >= alloc_size + sizeof(FreeBlock) + MIN_ALLOCATION_SIZE)
            {
                FreeBlock *new_block = (FreeBlock *)((char *)current + alloc_size);
                new_block->size = current->size - alloc_size;
                new_block->next = current->next;

                if (prev == NULL)
                {
                    allocator->free_list = new_block;
                }
                else
                {
                    prev->next = new_block;
                }
                current->size = alloc_size;
            }
            else
            {
                if (prev == NULL)
                {
                    allocator->free_list = current->next;
                }
                else
                {
                    prev->next = current->next;
                }
            }
```

```
            size_t *block_size = (size_t *)current;
            *block_size = current->size - sizeof(size_t);
            return (char *)current + sizeof(size_t);
        }

        prev = current;
        current = current->next;
    }

    return NULL;
}

void allocator_free(Allocator *const allocator, void *const memory)
{
    if (allocator == NULL || memory == NULL)
    {
        return;
    }

    size_t *block_size = (size_t *)((char *)memory - sizeof(size_t));
    FreeBlock *freed_block = (FreeBlock *)((char *)memory - sizeof(size_t));
    freed_block->size = *block_size + sizeof(size_t);

    FreeBlock *current = allocator->free_list;
    FreeBlock *prev = NULL;

    while (current != NULL && current < freed_block)
    {
        prev = current;
        current = current->next;
    }

    if (prev == NULL)
    {
        freed_block->next = allocator->free_list;
        allocator->free_list = freed_block;
    }
    else
    {
        freed_block->next = prev->next;
        prev->next = freed_block;
    }

    current = allocator->free_list;
    prev = NULL;
    while (current != NULL)
    {
        if (prev != NULL && (char *)prev + prev->size == (char *)current)
        {
            prev->size += current->size;
            prev->next = current->next;
            current = prev->next;
        }
```

```
        else
        {
            prev = current;
            current = current->next;
        }
    }
}
```

main.c

```
#include "allocator.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>

#define MEMORY_SIZE (1 << 20)
#define ALLOCATIONS 10000
#define MAX_ALLOC_SIZE 4096

typedef Allocator *(*allocator_create_func)(void *const memory, const size_t size);
typedef void (*allocator_destroy_func)(Allocator *const allocator);
typedef void *(*allocator_alloc_func)(Allocator *const allocator, const size_t size);
typedef void (*allocator_free_func)(Allocator *const allocator, void *const memory);

static void *sys_alloc(size_t size)
{
    return VirtualAlloc(NULL, size, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
}

static void sys_free(void *ptr, size_t size)
{
    VirtualFree(ptr, 0, MEM_RELEASE);
}

long long get_current_time_ns()
{
    LARGE_INTEGER count, freq;
    QueryPerformanceCounter(&count);
    QueryPerformanceFrequency(&freq);
    return (long long)((double)count.QuadPart / freq.QuadPart * 1000000000.0);
}

int main(int argc, char *argv[])
{
    void *memory = sys_alloc(MEMORY_SIZE);
    if (memory == NULL)
    {
        fprintf(stderr, "Failed to allocate memory\n");
        return 1;
    }

    allocator_create_func create_func = NULL;
    allocator_destroy_func destroy_func = NULL;
```

```
    allocator_alloc_func alloc_func = NULL;
    allocator_free_func free_func = NULL;

    HMODULE lib = NULL;

    if (argc > 1)
    {
        lib = LoadLibraryA(argv[1]);

        if (lib == NULL)
        {
            fprintf(stderr, "Failed to load library: %s\n", argv[1]);
            fprintf(stderr, "Error code: %lu\n", GetLastError());
        }
        else
        {
            create_func = (allocator_create_func)GetProcAddress(lib, "allocator_create");
            destroy_func = (allocator_destroy_func)GetProcAddress(lib,
"allocator_destroy");
            alloc_func = (allocator_alloc_func)GetProcAddress(lib, "allocator_alloc");
            free_func = (allocator_free_func)GetProcAddress(lib, "allocator_free");

            if (create_func == NULL || destroy_func == NULL || alloc_func == NULL ||
free_func == NULL)
            {
                fprintf(stderr, "Failed to get function addresses\n");
                FreeLibrary(lib);
                lib = NULL;
            }
        }
    }

    if (create_func == NULL)
    {
        create_func = (allocator_create_func)sys_alloc;
        destroy_func = (allocator_destroy_func)sys_free;
        alloc_func = (allocator_alloc_func)sys_alloc;
        free_func = (allocator_free_func)sys_free;
    }

    Allocator *allocator = create_func(memory, MEMORY_SIZE);
    if (allocator == NULL && lib != NULL)
    {
        fprintf(stderr, "Failed to create allocator\n");
        FreeLibrary(lib);
        sys_free(memory, MEMORY_SIZE);
        return 1;
    }

    void *allocations[ALLOCATIONS];
    size_t allocation_sizes[ALLOCATIONS];
    size_t total_allocated = 0;

    long long start_alloc = get_current_time_ns();
```

```
    for (int i = 0; i < ALLOCATIONS; i++)
    {
        size_t size = rand() % MAX_ALLOC_SIZE + 1;
        allocations[i] = alloc_func(allocator, size);
        if (allocations[i] != NULL)
        {
            allocation_sizes[i] = size;
            total_allocated += size;
        }
        else
        {
            allocation_sizes[i] = 0;
        }
    }
    long long end_alloc = get_current_time_ns();

    long long start_free = get_current_time_ns();
    for (int i = 0; i < ALLOCATIONS; i++)
    {
        if (allocations[i] != NULL)
        {
            free_func(allocator, allocations[i]);
        }
    }
    long long end_free = get_current_time_ns();

    if (allocator != NULL && create_func != (allocator_create_func)sys_alloc)
    {
        destroy_func(allocator);
    }

    if (lib != NULL)
    {
        FreeLibrary(lib);
    }

    if (create_func == (allocator_create_func)sys_alloc)
    {
        sys_free(memory, MEMORY_SIZE);
    }

    printf("Total allocated: %zu bytes\n", total_allocated);
    printf("Allocation time: %f seconds\n", (double)(end_alloc - start_alloc) /
1000000000.0);
    printf("Free time: %f seconds\n", (double)(end_free - start_free) / 1000000000.0);
    if (total_allocated > 0)
    {
        printf("Usage factor: %f\n", (double)total_allocated / MEMORY_SIZE);
    }
    else
    {
        printf("Usage factor: N/A (no successful allocations)\n");
    }
```

```
    return 0;
}
```

Протокол работы программы

Компиляция и запуск:

```
gcc -shared -o mckusick_karels.dll mckusick_karels.c "-Wl,--out-
implib,libmckusick_karels.a"
gcc -shared -o buddy.dll buddy.c "-Wl,--out-implib,libbuddy.a"
gcc main.c -o main.exe -L. -lmckusick_karels -lbuddy
```

Тестирование:

PS D:\code\osi\lab4> .\main.exe mckusick_karels.dll

Total allocated: 1042753 bytes

Allocation time: 0.000223 seconds

Free time: 0.000008 seconds

Usage factor: 0.994447

PS D:\code\osi\lab4> .\main.exe buddy.dll

Total allocated: 783549 bytes

Allocation time: 0.000525 seconds

Free time: 0.000037 seconds

Usage factor: 0.747251

PS D:\code\osi\lab4> .\main.exe

Total allocated: 0 bytes

Allocation time: 0.002842 seconds

Free time: 0.000008 seconds

Usage factor: N/A (no successful allocations)

nttrace:

buddy.log

Loaded DLL at 00007FF8B8110000 C:\Windows\SYSTEM32\ntdll.dll
NtAllocateVirtualMemoryEx(ProcessHandle=-1, lpAddress=0x7ff8b827dc38 [0x00007ff5d44c0000], ZeroBits=0x00000004785ff430, pSize=0x102000 [0], flAllocationType=4, DataBuffer=0x4785ff398, DataCount=1) => 0
NtAllocateVirtualMemoryEx(ProcessHandle=-1, lpAddress=0x7ff8b827dc30 [0x00007ff5d64c0000], ZeroBits=0x00000004785ff438, pSize=0x1000 [0], flAllocationType=4, DataBuffer=null, DataCount=0) => 0
NtAllocateVirtualMemoryEx(ProcessHandle=-1, lpAddress=0x7ff8b827dca0 [0x00007ff4d44a0000], ZeroBits=0x00000004785ff3e0, pSize=0x102000 [0], flAllocationType=4, DataBuffer=0x4785ff348, DataCount=1) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785feee0 [0x000001d86d360000], ZeroBits=0, pSize=0x4785feee8 [0x00180000], flAllocationType=0x2000, flProtect=4) => 0
NtFreeVirtualMemory(ProcessHandle=-1, lpAddress=0x4785feee0 [0x000001d86d360000], pSize=0x4785feed8 [0x00080000], flFreeType=0x8000) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785feec8 [0x000001d86d3e0000], ZeroBits=0, pSize=0x4785feec0 [0x2000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785fed30 [0x000001d86d3e2000], ZeroBits=0, pSize=0x4785fedd8 [0x2000], flAllocationType=0x1000, flProtect=4) => 0
NtFreeVirtualMemory(ProcessHandle=-1, lpAddress=0x4785ff510 [0x000001d86d300000], pSize=0x4785ff518 [0x00020000], flFreeType=0x8000) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785fe960 [0x000001d86d3e4000], ZeroBits=0, pSize=0x4785fea08 [0x1000], flAllocationType=0x1000, flProtect=4) => 0
Loaded DLL at 00007FF8B6FE0000 C:\Windows\System32\KERNEL32.DLL
NtMapViewOfSection(SectionHandle=0x5c, ProcessHandle=-1, BaseAddress=0x1d86d3e3e70 [0x00007ff8b6fe0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x1d86d3e3dd0 [0x000c2000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0
Loaded DLL at 00007FF8B5E30000 C:\Windows\System32\KERNELBASE.dll
NtMapViewOfSection(SectionHandle=0x40, ProcessHandle=-1, BaseAddress=0x1d86d3e4550 [0x00007ff8b5e30000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x1d86d3e44b0 [0x002fe000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0
NtCreateSection(SectionHandle=0x4785feab0 [0x5c], DesiredAccess=DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|0x1f, ObjectAttributes=null, SectionSize=0x4785feaa0 [65536], Protect=4, Attributes=0x08000000, FileHandle=0) => 0
NtMapViewOfSection(SectionHandle=0x40, ProcessHandle=-1, BaseAddress=0x4785feaa8 [0x00007ff4d43a0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x4785feab8 [0x00100000], InheritDisposition=2 [ViewUnmap], AllocationType=0x00500000, Protect=2) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785fe490 [0x000001d86d3e5000], ZeroBits=0, pSize=0x4785fe538 [0x1000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785fe3b0 [0x000001d86d3e6000], ZeroBits=0, pSize=0x4785fe458 [0x1000], flAllocationType=0x1000, flProtect=4) => 0
NtMapViewOfSection(SectionHandle=0x7c, ProcessHandle=-1, BaseAddress=0x4785feae0 [0x000001d86d310000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x4785feae8 [0x1000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0
NtMapViewOfSection(SectionHandle=0x80, ProcessHandle=-1, BaseAddress=0x4785feae0 [0x000001d86d360000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x4785feae8 [0x1000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0
NtMapViewOfSection(SectionHandle=0x84, ProcessHandle=-1, BaseAddress=0x4785feae0 [0x000001d86d370000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x4785feae8 [0x1000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785fe4f0 [0x000001d86d3e7000], ZeroBits=0, pSize=0x4785fe598 [0x1000], flAllocationType=0x1000, flProtect=4) => 0
Loaded DLL at 00007FF8B6250000 C:\Windows\System32\msvcrt.dll

NtMapViewOfSection(SectionHandle=0xa4, ProcessHandle=-1, BaseAddress=0x1d86d3e6f10 [0x00007ff8b6250000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x1d86d3e3dd0 [0x0009e000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785fec00 [0x000001d86d5b0000], ZeroBits=0, pSize=0x4785fec08 [0x00110000], flAllocationType=0x2000, flProtect=4) => 0
NtFreeVirtualMemory(ProcessHandle=-1, lpAddress=0x4785fec00 [0x000001d86d5b0000], pSize=0x4785febf8 [0x00100000], flFreeType=0x8000) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785febe8 [0x000001d86d6b0000], ZeroBits=0, pSize=0x4785febe0 [0x2000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785fea70 [0x000001d86d3e8000], ZeroBits=0, pSize=0x4785feb18 [0x2000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785fea40 [0x000001d86d6b2000], ZeroBits=0, pSize=0x4785feae8 [0x1000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785fea40 [0x000001d86d6b3000], ZeroBits=0, pSize=0x4785feae8 [0x1000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785fea20 [0x000001d86d6b4000], ZeroBits=0, pSize=0x4785feac8 [0x2000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785d8c40 [0x000001d86d5b0000], ZeroBits=0, pSize=0x4785d8c48 [0x00100000], flAllocationType=0x3000, flProtect=4) => 0
NtCreateSection(SectionHandle=0x4785d84f8 [0xac], DesiredAccess=0xd, ObjectAttributes=null, SectionSize=null, Protect=0x10, Attributes=0x01000000, FileHandle=0x88) => 0
Loaded DLL at 00007FF8A9D40000 D:\code\osi\lab4\buddy.dll
NtMapViewOfSection(SectionHandle=0xac, ProcessHandle=-1, BaseAddress=0x1d86d3e7ae0 [0x00007ff8a9d40000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x1d86d3e3dd0 [0x00014000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785d8c00 [0x000001d86d380000], ZeroBits=0, pSize=0x4785d8c08 [0x1000], flAllocationType=0x3000, flProtect=4) => 0
NtFreeVirtualMemory(ProcessHandle=-1, lpAddress=0x4785d8c18 [0x000001d86d380000], pSize=0x4785d8c10 [0x1000], flFreeType=0x8000) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x4785d82b0 [0x000001d86d3ea000], ZeroBits=0, pSize=0x4785d8358 [0x2000], flAllocationType=0x1000, flProtect=4) => 0
NtUnmapViewOfSection(ProcessHandle=-1, BaseAddress=0x7ff8a9d40000) => 0

mckusick_karels.log

Loaded DLL at 00007FF8B8110000 C:\Windows\SYSTEM32\ntdll.dll
NtAllocateVirtualMemoryEx(ProcessHandle=-1, lpAddress=0x7ff8b827dc38 [0x00007ff597f70000], ZeroBits=0x000000ab54bff270, pSize=0x102000 [0], flAllocationType=4, DataBuffer=0xab54bff1d8, DataCount=1) => 0
NtAllocateVirtualMemoryEx(ProcessHandle=-1, lpAddress=0x7ff8b827dc30 [0x00007ff599f70000], ZeroBits=0x000000ab54bff278, pSize=0x1000 [0], flAllocationType=4, DataBuffer=null, DataCount=0) => 0
NtAllocateVirtualMemoryEx(ProcessHandle=-1, lpAddress=0x7ff8b827dca0 [0x00007ff497f50000], ZeroBits=0x000000ab54bff220, pSize=0x102000 [0], flAllocationType=4, DataBuffer=0xab54bff188, DataCount=1) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfed20 [0x00000240002d0000], ZeroBits=0, pSize=0xab54bfed28 [0x002b0000], flAllocationType=0x2000, flProtect=4) => 0
NtFreeVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfed20 [0x00000240002d0000], pSize=0xab54bfed18 [0x001b0000], flFreeType=0x8000) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfed08 [0x0000024000480000], ZeroBits=0, pSize=0xab54bfed00 [0x2000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfeb70 [0x0000024000482000], ZeroBits=0, pSize=0xab54bfec18 [0x2000], flAllocationType=0x1000, flProtect=4) => 0
NtFreeVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bff350 [0x0000024000270000], pSize=0xab54bff358 [0x00020000], flFreeType=0x8000) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfe7a0 [0x0000024000484000], ZeroBits=0, pSize=0xab54bfe848 [0x1000], flAllocationType=0x1000, flProtect=4) => 0
Loaded DLL at 00007FF8B6FE0000 C:\Windows\System32\KERNEL32.DLL
NtMapViewOfSection(SectionHandle=0x5c, ProcessHandle=-1, BaseAddress=0x24000483e80 [0x00007ff8b6fe0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x24000483de0 [0x000c2000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0
Loaded DLL at 00007FF8B5E30000 C:\Windows\System32\KERNELBASE.dll
NtMapViewOfSection(SectionHandle=0x60, ProcessHandle=-1, BaseAddress=0x24000484560 [0x00007ff8b5e30000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x240004844c0 [0x002fe000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0
NtCreateSection(SectionHandle=0xab54bfe8f0 [0x64], DesiredAccess=DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|0x1f, ObjectAttributes=null, SectionSize=0xab54bfe8e0 [65536], Protect=4, Attributes=0x08000000, FileHandle=0) => 0
NtMapViewOfSection(SectionHandle=0x5c, ProcessHandle=-1, BaseAddress=0xab54bfe8e8 [0x00007ff497e50000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0xab54bfe8f8 [0x00100000], InheritDisposition=2 [ViewUnmap], AllocationType=0x00500000, Protect=2) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfe2d0 [0x0000024000485000], ZeroBits=0, pSize=0xab54bfe378 [0x1000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfe1f0 [0x0000024000486000], ZeroBits=0, pSize=0xab54bfe298 [0x1000], flAllocationType=0x1000, flProtect=4) => 0
NtMapViewOfSection(SectionHandle=0x7c, ProcessHandle=-1, BaseAddress=0xab54bfe920 [0x0000024000280000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0xab54bfe928 [0x1000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0
NtMapViewOfSection(SectionHandle=0x80, ProcessHandle=-1, BaseAddress=0xab54bfe920 [0x00000240003a0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0xab54bfe928 [0x1000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0
NtMapViewOfSection(SectionHandle=0x84, ProcessHandle=-1, BaseAddress=0xab54bfe920 [0x00000240003b0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0xab54bfe928 [0x1000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfe330 [0x0000024000487000], ZeroBits=0, pSize=0xab54bfe3d8 [0x1000], flAllocationType=0x1000, flProtect=4) => 0
Loaded DLL at 00007FF8B6250000 C:\Windows\System32\msvcrt.dll
NtMapViewOfSection(SectionHandle=0xa0, ProcessHandle=-1, BaseAddress=0x24000486f20 [0x00007ff8b6250000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x24000483de0 [0x0009e000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfea40 [0x0000024000580000], ZeroBits=0, pSize=0xab54bfea48 [0x00180000], flAllocationType=0x2000, flProtect=4) => 0
NtFreeVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfea40 [0x0000024000580000], pSize=0xab54bfea38 [0x00170000], flFreeType=0x8000) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfea28 [0x00000240006f0000], ZeroBits=0, pSize=0xab54bfea20 [0x2000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfe8b0 [0x0000024000488000], ZeroBits=0, pSize=0xab54bfe958 [0x2000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfe880 [0x00000240006f2000], ZeroBits=0, pSize=0xab54bfe928 [0x1000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfe880 [0x00000240006f3000], ZeroBits=0, pSize=0xab54bfe928 [0x1000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bfe860 [0x00000240006f4000], ZeroBits=0, pSize=0xab54bfe908 [0x2000], flAllocationType=0x1000, flProtect=4) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bd8a80 [0x0000024000580000], ZeroBits=0, pSize=0xab54bd8a88 [0x00100000], flAllocationType=0x3000, flProtect=4) => 0
NtCreateSection(SectionHandle=0xab54bd8338 [0xb0], DesiredAccess=0xd, ObjectAttributes=null, SectionSize=null, Protect=0x10, Attributes=0x01000000, FileHandle=0xa4) => 0
Loaded DLL at 00007FF8A9D40000 D:\code\osi\lab4\mckusick_karels.dll

NtMapViewOfSection(SectionHandle=0xb0, ProcessHandle=-1, BaseAddress=0x24000487c10 [0x00007ff8a9d40000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x24000487b60 [0x00014000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0
NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xab54bd80f0 [0x000002400048a000], ZeroBits=0, pSize=0xab54bd8198 [0x2000], flAllocationType=0x1000, flProtect=4) => 0
NtUnmapViewOfSection(ProcessHandle=-1, BaseAddress=0x7ff8a9d40000) => 0

main.log

Loaded DLL at 00007FF8B8110000 C:\Windows\SYSTEM32\ntdll.dll

NtAllocateVirtualMemoryEx(ProcessHandle=-1, lpAddress=0x7ff8b827dc38 [0x00007ff5779f0000], ZeroBits=0x000000d3199ff3a0, pSize=0x102000 [0], flAllocationType=4, DataBuffer=0xd3199ff308, DataCount=1) => 0

NtAllocateVirtualMemoryEx(ProcessHandle=-1, lpAddress=0x7ff8b827dc30 [0x00007ff5799f0000], ZeroBits=0x000000d3199ff3a8, pSize=0x1000 [0], flAllocationType=4, DataBuffer=null, DataCount=0) => 0

NtAllocateVirtualMemoryEx(ProcessHandle=-1, lpAddress=0x7ff8b827dca0 [0x00007ff4779d0000], ZeroBits=0x000000d3199ff350, pSize=0x102000 [0], flAllocationType=4, DataBuffer=0xd3199ff2b8, DataCount=1) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199fee50 [0x000001e709e50000], ZeroBits=0, pSize=0xd3199fee58 [0x00120000], flAllocationType=0x2000, flProtect=4) => 0

NtFreeVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199fee50 [0x000001e709e50000], pSize=0xd3199fee48 [0x00020000], flFreeType=0x8000) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199fee38 [0x000001e709e70000], ZeroBits=0, pSize=0xd3199fee30 [0x2000], flAllocationType=0x1000, flProtect=4) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199feca0 [0x000001e709e72000], ZeroBits=0, pSize=0xd3199fed48 [0x2000], flAllocationType=0x1000, flProtect=4) => 0

NtFreeVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199ff480 [0x000001e709df0000], pSize=0xd3199ff488 [0x00020000], flFreeType=0x8000) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199fe8d0 [0x000001e709e74000], ZeroBits=0, pSize=0xd3199fe978 [0x1000], flAllocationType=0x1000, flProtect=4) => 0

Loaded DLL at 00007FF8B6FE0000 C:\Windows\System32\KERNEL32.DLL

NtMapViewOfSection(SectionHandle=0x58, ProcessHandle=-1, BaseAddress=0x1e709e73e60 [0x00007ff8b6fe0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x1e709e73dc0 [0x000c2000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0

Loaded DLL at 00007FF8B5E30000 C:\Windows\System32\KERNELBASE.dll

NtMapViewOfSection(SectionHandle=0x5c, ProcessHandle=-1, BaseAddress=0x1e709e74540 [0x00007ff8b5e30000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x1e709e744a0 [0x002fe000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0

NtCreateSection(SectionHandle=0xd3199fea20 [0x64], DesiredAccess=DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|0x1f, ObjectAttributes=null, SectionSize=0xd3199fea10 [65536], Protect=4, Attributes=0x08000000, FileHandle=0) => 0

NtMapViewOfSection(SectionHandle=0x60, ProcessHandle=-1, BaseAddress=0xd3199fea18 [0x00007ff4778d0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0xd3199fea28 [0x00100000], InheritDisposition=2 [ViewUnmap], AllocationType=0x00500000, Protect=2) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199fe400 [0x000001e709e75000], ZeroBits=0, pSize=0xd3199fe4a8 [0x1000], flAllocationType=0x1000, flProtect=4) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199fe320 [0x000001e709e76000], ZeroBits=0, pSize=0xd3199fe3c8 [0x1000], flAllocationType=0x1000, flProtect=4) => 0

NtMapViewOfSection(SectionHandle=0x84, ProcessHandle=-1, BaseAddress=0xd3199fea50 [0x000001e709e00000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0xd3199fea58 [0x1000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0

NtMapViewOfSection(SectionHandle=0x88, ProcessHandle=-1, BaseAddress=0xd3199fea50 [0x000001e709e50000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0xd3199fea58 [0x1000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0

NtMapViewOfSection(SectionHandle=0x8c, ProcessHandle=-1, BaseAddress=0xd3199fea50 [0x000001e709e60000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0xd3199fea58 [0x1000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199fe460 [0x000001e709e77000], ZeroBits=0, pSize=0xd3199fe508 [0x1000], flAllocationType=0x1000, flProtect=4) => 0

Loaded DLL at 00007FF8B6250000 C:\Windows\System32\msvcrt.dll

NtMapViewOfSection(SectionHandle=0xb0, ProcessHandle=-1, BaseAddress=0x1e709e76ef0 [0x00007ff8b6250000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x1e709e73dc0 [0x0009e000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199feb70 [0x000001e70a040000], ZeroBits=0, pSize=0xd3199feb78 [0x00170000], flAllocationType=0x2000, flProtect=4) => 0

NtFreeVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199feb70 [0x000001e70a040000], pSize=0xd3199feb68 [0x00160000], flFreeType=0x8000) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199feb58 [0x000001e70a1a0000], ZeroBits=0, pSize=0xd3199feb50 [0x2000], flAllocationType=0x1000, flProtect=4) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199fe9e0 [0x000001e709e78000], ZeroBits=0, pSize=0xd3199fea88 [0x2000], flAllocationType=0x1000, flProtect=4) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199fe9b0 [0x000001e70a1a2000], ZeroBits=0, pSize=0xd3199fea58 [0x1000], flAllocationType=0x1000, flProtect=4) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199fe9b0 [0x000001e70a1a3000], ZeroBits=0, pSize=0xd3199fea58 [0x1000], flAllocationType=0x1000, flProtect=4) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199fe990 [0x000001e70a1a4000], ZeroBits=0, pSize=0xd3199fea38 [0x2000], flAllocationType=0x1000, flProtect=4) => 0

(ProcessHandle=-1, lpAddress=0xd3199d8bb0 [0], ZeroBits=0, pSize=0xd3199d8bb8 [0], flAllocationType=0x3000, flProtect=4) => 0xc000000d [87 '[illegible].']

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199d8bb0 [0], ZeroBits=0, pSize=0xd3199d8bb8 [0], flAllocationType=0x3000, flProtect=4) => 0xc000000d [87 '[illegible].']

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199d8bb0 [0], ZeroBits=0, pSize=0xd3199d8bb8 [0], flAllocationType=0x3000, flProtect=4) => 0xc000000d [87 '[illegible].']

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199d8bb0 [0], ZeroBits=0, pSize=0xd3199d8bb8 [0], flAllocationType=0x3000, flProtect=4) => 0xc000000d [87 '[illegible].']

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199d8bb0 [0], ZeroBits=0, pSize=0xd3199d8bb8 [0], flAllocationType=0x3000, flProtect=4) => 0xc000000d [87 '[illegible].']

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199d8bb0 [0], ZeroBits=0, pSize=0xd3199d8bb8 [0], flAllocationType=0x3000, flProtect=4) => 0xc000000d [87 '[illegible].']

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199d8bb0 [0], ZeroBits=0, pSize=0xd3199d8bb8 [0], flAllocationType=0x3000, flProtect=4) => 0xc000000d [87 '[illegible].']

...

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199d8bb0 [0], ZeroBits=0, pSize=0xd3199d8bb8 [0], flAllocationType=0x3000, flProtect=4) => 0xc000000d [87 '[illegible].']

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199d8bb0 [0], ZeroBits=0, pSize=0xd3199d8bb8 [0], flAllocationType=0x3000, flProtect=4) => 0xc000000d [87 '[illegible].']

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199d8bb0 [0], ZeroBits=0, pSize=0xd3199d8bb8 [0], flAllocationType=0x3000, flProtect=4) => 0xc000000d [87 '[illegible].']

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199d8bb0 [0], ZeroBits=0, pSize=0xd3199d8bb8 [0], flAllocationType=0x3000, flProtect=4) => 0xc000000d [87 '[illegible].']

NtFreeVirtualMemory(ProcessHandle=-1, lpAddress=0xd3199d8bb8 [0x000001e70a040000], pSize=0xd3199d8bb0 [0x00100000], flFreeType=0x8000) => 0

NtAllocateVirtualMemory: Выделение виртуальной памяти.

NtFreeVirtualMemory: Освобождение виртуальной памяти.

NtCreateSection: Создание объекта "секция". Секция - это объект ядра, представляющий собой область памяти, которую можно отобразить в адресное пространство одного или нескольких процессов.

NtMapViewOfSection: Отображение (mapping) секции в адресное пространство процесса.

NtUnmapViewOfSection: Отмена отображения (unmapping) секции из адресного пространства процесса.

Вывод

В ходе лабораторной работы были реализованы и протестированы два алгоритма управления памятью: метод Buddy System и алгоритм Мак-Кьюзика-Кэрелса. Buddy System продемонстрировал более высокую скорость выделения и освобождения блоков, но при этом имел тенденцию к большему количеству внутренней фрагментации. Алгоритм Мак-Кьюзика-Кэрелса показал себя более эффективным в плане использования памяти, минимизируя потери на фрагментацию, хотя и с несколько меньшей скоростью операций. Таким образом, выбор подходящего аллокатора зависит от приоритетов: скорости или эффективности использования памяти.