Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Мазепа Илья Алексеевич

Группа: М8О-209Б-23

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_

Москва, 2024

**GitHub репозиторий:** https://github.com/Tyhyqo/mai\_oc

# Тема

# Аллокаторы памяти

**Цель курсового проекта**

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

2. Проведение исследования в выбранной предметной области

**Задание**

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

## Введение

В данном курсовом проекте проведено исследование двух алгоритмов аллокации памяти:

1. Аллокатор на основе списка свободных блоков (First-Fit Free List Allocator).
2. Аллокатор на основе buddy-системы (блоки кратные степеням двойки).

Цель исследования – сравнить данные алгоритмы по следующим характеристикам:

* Фактор использования (эффективность использования памяти); - Скорость выделения блоков;
* Скорость освобождения блоков;
* Простота использования аллокатора.

Каждый из аллокаторов реализует функции, аналогичные стандартным malloc и free, и инициализируется заранее выделенными страницами памяти ядром. Разработана стратегия тестирования, минимизирующая неизбежные накладные расходы при измерении ключевых характеристик.

## 1. Описание алгоритмов

### 1.1 Аллокатор на основе списка свободных блоков

Принцип работы:

* Реализуется через поддержание связанного списка свободных блоков. - При вызове функции alloc производится обход списка свободных блоков и выбирается первый, достаточный по размеру.
* Если блок больше запрашиваемого размера, происходит разделение: выделяется нужная часть, а оставшийся фрагмент возвращается в список.
* При вызове free блок возвращается в список свободных блоков, после чего проверяется возможность объединения со смежными блоками (слияние).

Преимущества:

* Гибкость – поддержка произвольных размеров выделяемых блоков.
* Простота реализации.

Недостатки:

* Может возникать внешняя фрагментация.
* Время поиска подходящего блока может возрастать при большом количестве свободных блоков.

### 1.2 Buddy-система

Принцип работы:

* Память делится на блоки, размеры которых всегда являются степенью двойки.
* При выделении памяти определяется минимальный порядок (размер блока), удовлетворяющий запросу; если блок большего порядка, то он рекурсивно делится до нужного размера.
* При освобождении происходит попытка объединения (слияния) блоков-«бадди», если их сосед не занят.

Преимущества:

* Высокая скорость операций выделения и освобождения благодаря логарифмической структуре поиска.
* Меньшая внешняя фрагментация при объединении блоков.

Недостатки:

* Внутренняя фрагментация – возможно выделение большего объёма памяти, чем требуется, из-за кратности степеням двойки.
* Сложнее в реализации по сравнению с free list аллокатором.

## 2. Процесс тестирования

Для тестирования реализованы единичные тесты в файле main.c, где производится серия последовательных операций выделения и освобождения памяти для каждого из аллокаторов. Основные моменты тестирования:

* Инициализация: Каждый аллокатор инициализируется заранее выделенным блоком памяти стандартной функцией malloc.
* Измерение времени:
  + Используется функция clock() для измерения времени выделения памяти (N вызовов alloc) и освобождения памяти (N вызовов free\_mem).
  + Для каждого теста результаты выводятся отдельно для аллокатора free list и для buddy-системы.
* Массивы указателей: Результаты выделения сохраняются в массив для последующего освобождения.
* Метрики:
  + Время выделения блоков памяти;
  + Время освобождения блоков памяти.

## 3. Обоснование подхода тестирования

* Минимизация накладных расходов: за счёт проведения серии (NUM\_ITERATIONS) операций можно свести к минимуму влияние возможных накладных расходов на время измерений.
* Изолированность тестов: Каждый аллокатор тестируется отдельно, что позволяет сравнить их скорость независимо друг от друга.
* Реальная нагрузка: Множественные последовательные вызовы функций имитируют рабочую нагрузку и позволяют оценить, насколько эффективно аллокаторы справляются с частыми операциями выделения и освобождения памяти.
* Использование стандартных средств измерения времени: Применение clock() обеспечивает достаточную точность для сравнительных тестов в рамках данного проекта.

## 4. Результаты тестирования

*Пример результатов (зависит от конкретной реализации и нагрузки):*

* Free-list аллокатор:
  + Время выделения памяти: 0.000052 секунд
  + Время освобождения памяти: 0.000090 секунд
* Buddy-система:
  + Время выделения памяти: 0.000039 секунд
  + Время освобождения памяти: 0.000012 секунд

*Комментарий:* В зависимости от характеристик конкретного тестового сценария и конфликтов фрагментации может наблюдаться, что buddy-система выделяет память быстрее за счёт логарифмического поиска, но освобождение может быть немного медленнее или наоборот.

## 5. Заключение

В ходе работы по сравнению двух алгоритмов аллокации памяти были получены следующие выводы:

* Фактор использования:
  + Free-list аллокатор обладает более гибким подходом для работы с произвольными размерами блоков, однако может страдать от внешней фрагментации.
  + Buddy-система гарантирует быстрое выделение и объединение блоков, но страдает от внутренней фрагментации, что может приводить к избыточному использованию памяти.
* Скорость выделения блоков:
  + Buddy-система демонстрирует преимущество в скорости выделения за счёт логарифмического подхода, однако в сценариях с небольшим количеством свободных блоков разница может быть не столь заметна.
* Скорость освобождения блоков:
  + Free-list аллокатор, благодаря слиянию соседних блоков, может работать достаточно эффективно, хотя алгоритм слияния может требовать корректировки для оптимальной работы.
  + Buddy-система требует корректного хранения метаданных (порядка блоков) для обеспечения правильного объединения, что делает процесс освобождения более сложным, но всё же быстрым за счёт применения битовых операций.
* Простота использования:
  + Реализация Free-list аллокатора проще и лучше подходит для динамичных сценариев, где размеры запросов сильно варьируются.
  + Buddy-система имеет более жёсткие ограничения (только кратные степеням двойки), что усложняет использование, но обеспечивает лучшую структурированность и скорость в сценариях с частыми операциями выделения/освобождения.

**Вывод:**

Выбор между двумя подходами зависит от конкретных требований к системе: если важна гибкость и динамичность — может быть предпочтён free-list аллокатор, а если критична скорость операций и объединение блоков — buddy-система окажется оптимальной.