МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 "Информационные технологий и прикладная математика"

Кафедра 806 "Вычислительная математика и программирование"

**Курсовой проект**

**по курсу "Операционные системы"**

**за 3 семестр**

Работу выполнил(а) студент(ка) группы М8О-206Б-20

Фурлетова Дарья Александровна, № по списку: 24

Контакты: e-mail: dafurletova@mail.ru

Работа сдана: 29 декабря 2021 г.

Преподаватель: Соколов А.А.

Итоговая оценка: \_\_\_\_\_.

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_.

Москва, 2021 г.

1. **Тема.**

Сравнение алгоритмов аллокации памяти.

1. **Цель курсового проекта.**

Целью является приобретение практических навыков в:

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

1. **Задание.**

Необходимо реализовать два алгоритма аллокации памяти и сравнить их. Каждый аллокатор памяти должен иметь функции аналогичные стандартным функциям free и malloc. Перед работой каждый аллокатор инициализируется свободными страницами памяти, выделенными стандартными средствами ядра. В отчёте необходимо отобразить следующее:

* Подробное описание каждого из исследуемых алгоритмов.
* Процесс тестирования.
* Обоснование подхода тестирования.
* Результат тестирования.
* Заключение по проведённой работе.

Каждый аллокатор должен обладать следующим интерфейсом:

* Allocator\* createMemoryAllocator(void\* realMemory, size\_t memory\_size) - создание аллокатор 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.

**Вариант № 21.**

Сравнить алгоритм аллокации на списках свободных блоков (находящий наиболее подходящий блок) и алгоритм двойников.

1. **Основная часть.**
   1. **Общая информация об аллокаторах.**

Аллокатор — специализированный класс, реализующий и инкапсулирующий малозначимые (с прикладной точки зрения) детали распределения и освобождения ресурсов компьютерной памяти. Стандартные функции malloc и free на самом деле имеют много проблем:

* Выделение нескольких байтов с помощью malloc происходит точно так же, как и выделение нескольких мегабайтов. В расчёт не берётся информация о том, что это за данные, где они будут располагаться и какой у них будет цикл жизни.
* Выделение памяти при помощи стандартных библиотечных функций или операторов обычно требует обращений к ядру операционной системы. Это может сказываться на производительности приложения.
* Они приводят к фрагментации кучи — состоянию, при котором информация в памяти разбросана в разных, не идущих последовательно блоках, из—за чего даже при достаточном суммарном объёме памяти возможна такая ситуация, что выделить блок в памяти для размещения информации будет невозможно.
* Плохая локальность указателей. Нет никакого способа узнать, какое именно место в память выделит вам malloc. Это может привести к тому, что будет происходить больше дорогостоящих промахов в кеше.
  1. **Аллокатор на списках свободных блоков.**

Суть аллокатора на списках свободных блоков заключается в том, что при каждом запросе на выделение памяти из доступной аллокатору памяти выделяется блока запрашиваемого размера (если это возможно). В конечном итоге вся память аллокатора будет разделена на список подряд идущих блоков, некоторые из которых будут заняты, а некоторые свободны.

Вся память аллокатора разделена на блоки, каждый из которых содержит заголовок. В заголовке представлена информация о размере этого блока, размере предыдущего блока и логическая переменная, показывающая свободен ли блок. При создании такого аллокатора сразу выделяется память запрашиваемого размера, в которой создаётся один блок. При запросе на выделение памяти определённого размера ищется наиболее подходящий свободный блок, из которого выделяется блок запрашиваемого размера. При деаллокации ранее выделенной памяти сначала проверяется валидность переданного указателя — если пользователь хочет его вернуть аллокатору, значит этот указатель должен был быть когда-то этим же аллокатором выдан. После просто в заголовке меняем информацию о том, что блок доступен для использования.

Важной деталью этого аллокатора является возможность дефрагментации. При деаллокации определённого блока памяти происходит проверка доступности соседних блоков — если какой-то из этих блоков доступен, то происходит слияние текущего блока со свободным соседним.

Листинг этой реализации аллокатора приведён в файлах “free\_list.h” и “free\_list.cpp”.

* 1. **Алгоритм двойников**

Алгоритм двойников - это алгоритм распределения памяти, который делит память на разделы, чтобы попытаться удовлетворить запрос памяти как можно более подходящим образом. Эта система использует разделение памяти на половины, чтобы попытаться обеспечить наилучшее соответствие.

Каждый блок разделяется на два меньших блока. Каждый блок памяти в этой системе имеет порядок, он представляет собой целое число в диапазоне от 0 до заданного верхнего предела. Размер блока порядка n пропорционален , так что блоки текущего уровня ровно в два раза больше блоков, которые на порядок ниже. Размеры блоков степени двойки упрощают вычисление адресов, поскольку все двойники выровнены по границам адресов памяти, равным степеням двойки. Когда больший блок разделяется, он делится на два меньших блока, и каждый меньший блок становится уникальным дополнением к другому. Разделенный блок может быть объединен только с его уникальным блоком-двойником, который затем преобразует больший блок, из которого они были разделены.

Листинг этой реализации аллокатора приведён в файлах “buddy\_alloc.h” и “buddy\_alloc.cpp”.

1. **Тестирование.**

Тестирование производилось с помощью класса Benchmark и библиотеки chrono, из которой использовались функции для нахождения времени начала и конца работы программы. Memory peak – максимально используемая память.

Для списка свободных блоков:

FREE LIST  
BENCHMARK: ALLOCATION  
Size: 32  
RESULTS:  
Operations: 5000  
Time elapsed: 29 ms  
Op per sec: 172.414 ops/ms  
Timer per op: 0.0058 ms/ops  
Memory peak: 160024 bytes  
  
BENCHMARK: ALLOCATION  
Size: 64  
RESULTS:  
Operations: 5000  
Time elapsed: 28 ms  
Op per sec: 178.571 ops/ms  
Timer per op: 0.0056 ms/ops  
Memory peak: 320024 bytes  
  
BENCHMARK: ALLOCATION  
Size: 256  
RESULTS:  
Operations: 5000  
Time elapsed: 34 ms  
Op per sec: 147.059 ops/ms  
Timer per op: 0.0068 ms/ops  
Memory peak: 1280024 bytes  
  
BENCHMARK: ALLOCATION  
Size: 512  
RESULTS:  
Operations: 5000  
Time elapsed: 33 ms  
Op per sec: 151.515 ops/ms  
Timer per op: 0.0066 ms/ops  
Memory peak: 2560024 bytes  
  
BENCHMARK: ALLOCATION  
Size: 1024  
RESULTS:  
Operations: 5000  
Time elapsed: 36 ms  
Op per sec: 138.889 ops/ms  
Timer per op: 0.0072 ms/ops  
Memory peak: 5120024 bytes  
  
BENCHMARK: ALLOCATION  
Size: 2048  
RESULTS:  
Operations: 5000  
Time elapsed: 43 ms  
Op per sec: 116.279 ops/ms  
Timer per op: 0.0086 ms/ops  
Memory peak: 10240024 bytes  
  
BENCHMARK: ALLOCATION  
Size: 4096  
RESULTS:  
Operations: 5000  
Time elapsed: 57 ms  
Op per sec: 87.7193 ops/ms  
Timer per op: 0.0114 ms/ops  
Memory peak: 20480024 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 32  
RESULTS:  
Operations: 5000  
Time elapsed: 23 ms  
Op per sec: 217.391 ops/ms  
Timer per op: 0.0046 ms/ops  
Memory peak: 160024 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 64  
RESULTS:  
Operations: 5000  
Time elapsed: 27 ms  
Op per sec: 185.185 ops/ms  
Timer per op: 0.0054 ms/ops  
Memory peak: 320024 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 256  
RESULTS:  
Operations: 5000  
Time elapsed: 31 ms  
Op per sec: 161.29 ops/ms  
Timer per op: 0.0062 ms/ops  
Memory peak: 1280024 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 512  
RESULTS:  
Operations: 5000  
Time elapsed: 35 ms  
Op per sec: 142.857 ops/ms  
Timer per op: 0.007 ms/ops  
Memory peak: 2560024 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 1024  
RESULTS:  
Operations: 5000  
Time elapsed: 34 ms  
Op per sec: 147.059 ops/ms  
Timer per op: 0.0068 ms/ops  
Memory peak: 5120024 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 2048  
RESULTS:  
Operations: 5000  
Time elapsed: 40 ms  
Op per sec: 125 ops/ms  
Timer per op: 0.008 ms/ops  
Memory peak: 10240024 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 4096  
RESULTS:  
Operations: 5000  
Time elapsed: 59 ms  
Op per sec: 84.7458 ops/ms  
Timer per op: 0.0118 ms/ops  
Memory peak: 20480024 bytes

Для алгоритма двойников:

BUDDY  
BENCHMARK: ALLOCATION  
Size: 32  
RESULTS:  
Operations: 5000  
Time elapsed: 0 ms  
Op per sec: inf ops/ms  
Timer per op: 0 ms/ops  
Memory peak: 160000 bytes  
  
BENCHMARK: ALLOCATION  
Size: 64  
RESULTS:  
Operations: 5000  
Time elapsed: 0 ms  
Op per sec: inf ops/ms  
Timer per op: 0 ms/ops  
Memory peak: 320000 bytes  
  
BENCHMARK: ALLOCATION  
Size: 256  
RESULTS:  
Operations: 5000  
Time elapsed: 0 ms  
Op per sec: inf ops/ms  
Timer per op: 0 ms/ops  
Memory peak: 1280000 bytes  
  
BENCHMARK: ALLOCATION  
Size: 512  
RESULTS:  
Operations: 5000  
Time elapsed: 1 ms  
Op per sec: 5000 ops/ms  
Timer per op: 0.0002 ms/ops  
Memory peak: 2560000 bytes  
  
BENCHMARK: ALLOCATION  
Size: 1024  
RESULTS:  
Operations: 5000  
Time elapsed: 1 ms  
Op per sec: 5000 ops/ms  
Timer per op: 0.0002 ms/ops  
Memory peak: 5120000 bytes  
  
BENCHMARK: ALLOCATION  
Size: 2048  
RESULTS:  
Operations: 5000  
Time elapsed: 1 ms  
Op per sec: 5000 ops/ms  
Timer per op: 0.0002 ms/ops  
Memory peak: 10240000 bytes  
  
BENCHMARK: ALLOCATION  
Size: 4096  
RESULTS:  
Operations: 5000  
Time elapsed: 1 ms  
Op per sec: 5000 ops/ms  
Timer per op: 0.0002 ms/ops  
Memory peak: 20480000 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 32  
RESULTS:  
Operations: 5000  
Time elapsed: 0 ms  
Op per sec: inf ops/ms  
Timer per op: 0 ms/ops  
Memory peak: 160000 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 64  
RESULTS:  
Operations: 5000  
Time elapsed: 0 ms  
Op per sec: inf ops/ms  
Timer per op: 0 ms/ops  
Memory peak: 320000 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 256  
RESULTS:  
Operations: 5000  
Time elapsed: 0 ms  
Op per sec: inf ops/ms  
Timer per op: 0 ms/ops  
Memory peak: 1280000 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 512  
RESULTS:  
Operations: 5000  
Time elapsed: 0 ms  
Op per sec: inf ops/ms  
Timer per op: 0 ms/ops  
Memory peak: 2560000 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 1024  
RESULTS:  
Operations: 5000  
Time elapsed: 1 ms  
Op per sec: 5000 ops/ms  
Timer per op: 0.0002 ms/ops  
Memory peak: 5120000 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 2048  
RESULTS:  
Operations: 5000  
Time elapsed: 1 ms  
Op per sec: 5000 ops/ms  
Timer per op: 0.0002 ms/ops  
Memory peak: 10240000 bytes  
  
BENCHMARK: ALLOCATION/FREE  
Size: 4096  
RESULTS:  
Operations: 5000  
Time elapsed: 1 ms  
Op per sec: 5000 ops/ms  
Timer per op: 0.0002 ms/ops  
Memory peak: 20480000 bytes

1. **Листинг программы.**

Продемонстрирован во время сдачи.

1. **Сравнение алгоритмов аллокации.**

Для сравнения алгоритмов аллокации будем замерять общее время работы аллокатора по аллокации / деаллокации блоков памяти. Тестовый набор содержит 5000 запросов на аллокацию / деаллокацию. Общая память аллокатора была порядка 1е8. На основе тестирования были получены следующие показатели:

1. Аллокатор на списке свободных блоков: 40-60 мс.
2. Аллокатор на алгоритме двойников: 0-2 мс.
3. **Вывод.**

По результатам тестирования можно увидеть, что алгоритм двойников, построенный на дереве отрезков является одним из самых быстрых алгоритмов, так как аллокация памяти и её очищение происходит за время порядка O (log n). Наверное, основным его недостатком является выделение 2\*n памяти для хранения информации о всех блоках памяти. Этот алгоритм легко дефрагментирует память рядом находящихся блоков, объединяя их в большие блоки. Также благодаря ДО не нужно хранить никакую информацию о размерах блока, так как она легко получается из смещения от начала памяти аллокатора.