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-211Б-23

Студент: Бугренков В.П. Преподаватель: Бахарев В.Д. Оценка:

Дата: 22.12.24

Москва, 2024

# Постановка задачи

Вариант 4.

Цель работы:

Приобретение практических навыков в: 1) Создании аллокаторов памяти и их анализу; 2) Создании динамических библиотек и программ, использующие динамические библиотеки.

Задание

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. Каждый аллокатор должен обладать следующим интерфейсом (могут быть отличия в зависимости от особенностей алгоритма):

Allocator\* allocator\_create(void \*const memory, const size\_t size) (инициализация аллокатора на памяти memory размера size);

void allocator\_destroy(Allocator \*const allocator) (деинициализация структуры аллокатора);

void\* allocator\_alloc(Allocator \*const allocator, const size\_t size) (выделение памяти аллокатором памяти размера size);

void allocator\_free(Allocator \*const allocator, void \*const memory) (возвращает выделенную память аллокатору);

Необходимо реализовать:

4. Алгоритм Мак-Кьюзика-Кэрелса и блоки по 2^n;

# Общий метод и алгоритм решения

# Использованные системные вызовы:

# mmap() – для выделения блока памяти (аналог malloc), используется для резервирования памяти для аллокатора.

# munmap() – для освобождения ранее выделенной памяти (аналог free),

# применяется для очистки ресурсов, выделенных через mmap().

# dlopen() – для загрузки динамической библиотеки во время выполнения программы, используется для подключения аллокатора из внешней библиотеки.

# dlsym() – для получения указателей на функции из динамически загруженной библиотеки, позволяет использовать функции аллокатора.

# dlclose() – для закрытия загруженной динамической библиотеки, освобождает ресурсы, связанные с загрузкой библиотеки.Реализованы аллокаторы:

# Аллокатор McKusick-Karels: аллокатор использует стратегию распределения памяти, где блоки памяти организуются в свободные списки, что позволяет эффективно управлять памятью и избежать фрагментации. Он использует списки свободных блоков и делит их на разные группы по размеру.

# Аллокатор блоков 2^n использует стратегию, где память делится на блоки кратные степеням двойки (например, 16, 32, 64 байт). Для каждого размера блоков существует список свободных блоков (free list). При выделении памяти запрашиваемый размер округляется до ближайшего 2^n, и блок извлекается из соответствующего списка. Если подходящий блок отсутствует, используется блок большего размера, который расщепляется на два меньших. Освобожденные блоки возвращаются в списки, и если соседние блоки свободны, они объединяются, уменьшая фрагментацию. Эта стратегия обеспечивает быстрое выделение, эффективное управление памятью, но может страдать от внутренней фрагментации, если размер запрашиваемой памяти меньше выделенного блока.

# Код программы

**parent.c**

#include <stdio.h>  
*#include* <stdlib.h>  
*#include* <unistd.h>

# Протокол работы программы

qwental@DESKTOP-NKF1EUK:~/workspace/OS\_LABS/lab3/src$ gcc child.c -o child

Division result is: 6.250000

**Strace:**

qwental@DESKTOP-NKF1EUK:~/workspace/OS\_LABS/lab3/src$ strace ./parent test.txt

+++ exited with 0 +++

# Вывод

В рамках лабораторной работы была разработана программа, демонстрирующая работу аллокатора передаваемого в качестве аргумента при вызове программы. Было реализовано 2 аллокатора и проведена работа по сравнениб их работоспособности.