Лабораторная работа №4: *аллокаторы памяти*

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);

В отчете необходимо отобразить следующее:

- Подробное описание каждого из исследуемых алгоритмов
- Процесс тестирования
- Обоснование подхода тестирования
- Результаты тестирования

Варианты

1. Списки свободных блоков (первое подходящее) и блоки по 2^n;
2. Списки свободных блоков (первое подходящее) и алгоритм Мак-Кьюзи-Кэрелса;
3. Списки свободных блоков (первое подходящее) и алгоритм двойников;
4. Алгоритм Мак-Кьюзика-Кэрелса и блоки по 2^n;
5. Алгоритм Мак-Кьюзика-Кэрелса и алгоритм двойников;
6. Блоки по 2^n и алгоритм двойников;
7. Списки свободных блоков (наиболее подходящее) и блоки по 2^n;
8. Списки свободных блоков (наиболее подходящее) и алгоритм Мак-Кьюзика-Кэрелса;
9. Списки свободных блоков (наиболее подходящее) и алгоритм двойников;