Курс “C++ Basic”

Домашнее задание “Многопоточный “взлом”

Некоторые вещи просто сделать,
но сложно обратить:
например, разбить яйцо

В этом задании мы напишем многопоточное приложение на основе **`std::thread`** и в качестве бонуса на практике убедимся, что не следует использовать в качестве электронной подписи документов хеш-функции, не входящие в подмножество специально с этой целью разработанных и криптографически стойких.

Требуется с помощью многопоточности ускорить программу, которая добавляет в конец входного файла строку `"He-he-he"` и произвольные четыре байта, подобранные так, чтобы CRC32 хеш полученного нового файла совпал в точности с CRC32 исходного.

Описание

В примере дана завершенная программа, которая запускается командой `./crc32_crack <путь к входному файлу> <путь к выходному файлу>` и записывает в выходной файл данные из исходного, к которым добавлена дополнительная строка и некоторые “мусорные” символы. Логика работы функции **`hack`** проста: в цикле осуществляется полный перебор всех возможных комбинаций четырех дополнительных байт до тех пор, пока не будет найден набор значений, при которых общая хеш-сумма совпадает с хешем исходного файла. За вычисления хэш-суммы отвечает функция **`crc32`**, модифицировать или дорабатывать ее в рамках работы не требуется.

Внимание! Предложенная реализация крайне не оптимальна (см. доп. задание 1), поэтому рекомендуется тестировать решения на файлах небольшого объема (~ до 100 байт).

Задание

Ускорить работу исходного примера, обеспечив параллельный многопоточный перебор возможных вариантов дополнительных четырех байт. Для этого следует разбить диапазон их возможных значений на **`t`** поддиапазонов и запускать вычисление хеш-функций независимо для каждого поддиапазона. Конкретное значение **`t`** можно принять равным **`std::thread::hardware_concurrency()`** (а можно действовать более гибко, см. доп. задание 2).

Внимание! Избегайте передачи в функцию потока разделяемых модифицируемых данных. В рамках задания не предполагается дополнительной синхронизации работы потоков, кроме ожидания их завершения (**`std::thread::join()`**).

Ожидается, что после доработки исходный код программы будет компилироваться и запускаться командой `./crc32_crack <путь к входному файлу> <путь к выходному`

файлу>, а в результате работы программы будет создан новый файл с добавленными данными и хешем CRC32, совпадающим с исходным.

Дополнительные задания

Дополнительные задания не обязательны для сдачи работы, но их выполнение – хороший способ узнать что-то новое.

1. Оптимизация

Сложность 2/5

Перед ускорением существующей реализации с привлечением распараллеливания всегда стоит задуматься, исчерпаны ли иные возможные оптимизации. В исходном примере в файле **`main.cpp`** происходят многократные избыточные вычисления, которые “съедают” почти все время работы приложения. В рамках этого доп. задания предлагается найти и исправить этот недостаток. Хорошей идеей будет начать с внимательного изучения описании функции **`crc32`** в файле **`CRC32.hpp`**.

2. Динамическая балансировка

Сложность 5/5

В зависимости от того, какие приложения работают параллельно с нашим, запускать **`std::thread::hardware_concurrency()`** потоков может быть как удачной идеей, так и не очень. Предлагается попробовать непосредственно в ходе выполнения приложения определить оптимальное количество потоков для вычисления CRC32. Сравнительно простой способ это сделать - попробовать последовательно вычислить некоторое ограниченное количество хешей в 1, 2, 4 … потоков, замеряя в каждом из вариантов общее количество затраченного времени. Далее на основании полученного упрощенного динамического профиля – картины, с какой скоростью работает программа – выбрать количество потоков для расчета оставшихся хешей.