Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет
«Высшая школа экономики»

Кафедра «Компьютерная безопасность»

ОТЧЕТ
К ЛАБОРАТОРНОЙ РАБОТЕ №15

по дисциплине

«Языки программирования»

Работу выполнил
студент группы СКБ-222 _____ А.С. Вагин
подпись, дата

Работу проверил _____ С.А. Булгаков
подпись, дата

Москва 2023

Содержание

Постановка задачи

Разработать класс реализующий пул потоков.

В основной функции продемонстрировать работу класса на основе задания лабораторной работы №14.

Основная часть

1 Описание изменений

Была изменена функция *ParseChecksum*. Ее основную работу теперь выполняет функция *run* класса *ThreadPool*. Функционал измененной функции теперь отвечает за инициализацию *ThreadPool* и добавление туда заданий.

2 Класс ThreadPool

Был разработан класс *ThreadPool*, отвечающий за параллельное выполнение заданий. Его функционал ограничивается задачами лабораторной работы №14.

2.1 Добавление задач

Функция *addTask* принимает на вход структуру вида *task*. В виду реализации функция ничего не возвращает. Вывод осуществляет непосредственно сам поток в функции *run*.

3 Внутреннее хранилище задач

Задачи хранятся в массива типа *query* (FIFO), что позволяет эффективно распределять их между потоками. При любых взаимодействиях с массивом, доступ блокируется при помощи *std::mutex*, во избежание ошибок.

4 Работа потока

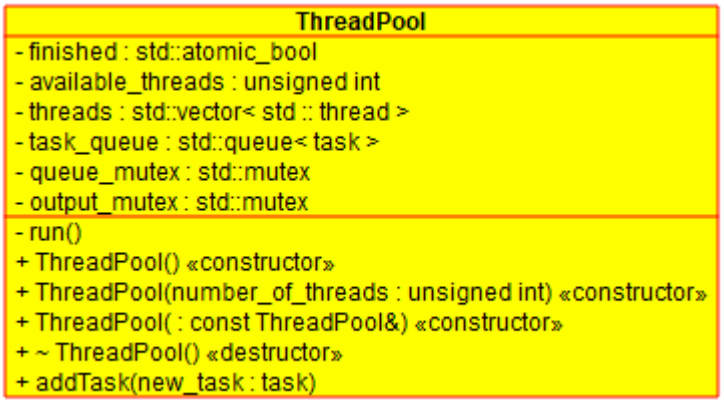
Потоки работают с создания объекта *ThreadPool* и до его удаления. Если внутренний массив задач не пуст, то поток вынимает оттуда *task*, после чего начинает его обрабатывать. При получении результата, поток выводит полученные данные, предварительно заблокировав вывод при помощи **std::mutex**.

5 Удаление *ThreadPool*

При вызове деструктора, объект дожидается, когда внутренний массив задач окажется пустым, после чего меняет значение переменной *finished* на **true**. В таком случае, потоки заканчивают оставшиеся задачи и завершаются, после чего объект и все внутренние потоки будут удалены.

Приложение А

А.1 UML-диаграмма *ThreadPool*



Приложение B

B.1 Код файла *ThreadPool*

```
#include <atomic>
#include <future>
#include <thread>
#include <mutex>
#include <vector>
#include <queue>
#include "hash.hpp"
#include "all_structures.hpp"


class ThreadPool {
    private:
        std::atomic_bool finished;
        unsigned int available_threads;
        std::vector<std::thread> threads;
        std::queue<task> task_queue;
        std::mutex queue_mutex;
        std::mutex output_mutex;

        void run() {
            while (!finished) {
                std::unique_lock<std::mutex> queue_lock(queue_mutex);
                if (!task_queue.empty()) {
                    task workingTask = std::move(task_queue.front());
                    task_queue.pop();
                    queue_lock.unlock();
                    auto bytes = getBytesFromFile(workingTask.filename);
                    std::string result = HashFactory::hash(bytes.data(),
                    bytes.size(), workingTask.algorithm);
                    output_mutex.lock();
                    if (workingTask.expected_hash == "") {
                        std::cout << "File " << workingTask.filename <<
                        " hash (" << algorithmToText(workingTask.algorithm) << "): " << std::endl;
                        std::cout << result << std::endl << std::endl;
                    } else if (workingTask.expected_hash == result) {
                        std::cout << "File " << workingTask.filename <<
                        " is complete! (" << algorithmToText(workingTask.algorithm) << ")" << std::endl << s
                    } else {
                        std::cout << "File " << workingTask.filename <<
                        " is corrupted! (" << algorithmToText(workingTask.algorithm) << ")" << std::endl;
                        std::cout << "Expected hash: " << workingTask.expected_hash << std::endl;
                        std::cout << "Got hash: " << result << std::endl << std::endl;
                    }
                    output_mutex.unlock();
                }
            }
        }
        public:
            ThreadPool(): finished(false) {
                available_threads = std::thread::hardware_concurrency();
                threads.reserve(available_threads);
                for (unsigned int i = 0; i < available_threads; ++i) {
                    threads.emplace_back(&ThreadPool::run, this);
                }
            }
            ThreadPool(unsigned int number_of_threads): finished(false),
            available_threads(number_of_threads) {
                threads.reserve(available_threads);
                for (unsigned int i = 0; i < available_threads; ++i) {
```

```
                threads.emplace_back(&ThreadPool::run, this);
            }
        }
        ThreadPool(const ThreadPool &) = delete;
        ThreadPool(ThreadPool &&) = delete;
        ~ThreadPool() {
            while (!task_queue.empty()) {}
            finished = true;
            for (auto& i : threads) {
                i.join();
            }
        }

        void addTask(task new_task) {
            queue_mutex.lock();
            task_queue.push(new_task);
            queue_mutex.unlock();
        }
};
```

B.2 Код файла *main.cpp*

```
#include <iostream>
#include <fstream>
#include <filesystem>
#include <string>
#include <future>
#include <mutex>
#include <exception>
#include "hash.hpp"
#include "filetools.cpp"
#include "thread_pool.hpp"


std::mutex output;

// Compare Hashes with expected or output
void comparingHashes(std::map<Hashes, std::vector<file>> files) {
    std::vector<Hashes> all_hashes { Hashes::MD5, Hashes::CRC32, Hashes::SHA256 };
    std::vector<std::future<std::string>> futures;
    ThreadPool thread_pool;
    std::string got_string;

    for (Hashes algorithm : all_hashes) {
        for (file current : files[algorithm]) {
            thread_pool.addTask(task(current, algorithm));
        }
    }
}

int main(int argc, const char** argv) {
    if (std::filesystem::exists("Checksum.ini")) {
        comparingHashes(parseChecksum("Checksum.ini"));
    }

    if (argc < 4 && !std::filesystem::exists("Checksum.ini")) std::cout <<
    "Usage: " << argv[0] << " filenames -a (crc32/md5/sha256)" << std::endl;
    else if (argc >= 4) {
        comparingHashes(parseArguments(argc, argv));
    }

    return 0;
}
```