#include <iostream>

#include <fstream>

#include <vector>

#include <thread>

#include <mutex>

#include <string>

#include <algorithm>

#include <memory>

using namespace std;

// Структура для хранения данных задачи

struct FileTask {

string input\_file;

string output\_file;

char char\_to\_remove;

int result;

};

// Глобальный мьютекс для синхронизации вывода

mutex cout\_mutex;

// Функция обработки файла

void process\_file(FileTask& task) {

ifstream in\_file(task.input\_file, ios::binary);

ofstream out\_file(task.output\_file, ios::binary);

// Проверка открытия файлов

if (!in\_file.is\_open()) {

lock\_guard<mutex> lock(cout\_mutex);

cerr << "Error: Could not open input file: " << task.input\_file << endl;

task.result = -1;

return;

}

if (!out\_file.is\_open()) {

lock\_guard<mutex> lock(cout\_mutex);

cerr << "Error: Could not open output file: " << task.output\_file << endl;

task.result = -1;

return;

}

int count = 0;

string buffer;

char ch;

// Чтение и обработка файла посимвольно

while (in\_file.get(ch)) {

if (ch != task.char\_to\_remove) {

out\_file.put(ch);

} else {

count++;

}

}

// Проверка ошибок чтения/записи

if (!in\_file.eof() || !out\_file.good()) {

lock\_guard<mutex> lock(cout\_mutex);

cerr << "Error: File processing failed for: " << task.input\_file << endl;

task.result = -1;

return;

}

task.result = count;

// Вывод информации о выполненной работе

lock\_guard<mutex> lock(cout\_mutex);

cout << "Processed " << task.input\_file

<< ": removed " << count << " occurrences of '"

<< task.char\_to\_remove << "'" << endl;

}

int main(int argc, char\* argv[]) {

// Проверка аргументов командной строки

if (argc < 4 || (argc - 1) % 3 != 0) {

cerr << "Usage: " << argv[0]

<< " <input1> <output1> <char1> [<input2> <output2> <char2> ...]" << endl;

return EXIT\_FAILURE;

}

// Создание задач для потоков

int num\_tasks = (argc - 1) / 3;

vector<FileTask> tasks(num\_tasks);

for (int i = 0; i < num\_tasks; ++i) {

tasks[i].input\_file = argv[1 + i\*3];

tasks[i].output\_file = argv[2 + i\*3];

// Проверка что передан ровно один символ

string char\_arg = argv[3 + i\*3];

if (char\_arg.size() != 1) {

cerr << "Error: Character argument must be a single character, got: "

<< char\_arg << endl;

return EXIT\_FAILURE;

}

tasks[i].char\_to\_remove = char\_arg[0];

tasks[i].result = 0;

}

// Создание и запуск потоков

vector<unique\_ptr<thread>> threads;

threads.reserve(num\_tasks);

for (auto& task : tasks) {

threads.emplace\_back(make\_unique<thread>(process\_file, ref(task)));

}

// Ожидание завершения всех потоков

for (auto& t : threads) {

t->join();

}

// Вывод итоговой информации

cout << "\nProcessing results:" << endl;

int total\_removed = 0;

int failed\_tasks = 0;

for (const auto& task : tasks) {

if (task.result == -1) {

cerr << "Failed: " << task.input\_file << endl;

failed\_tasks++;

} else {

cout << task.input\_file << ": removed " << task.result << " characters" << endl;

total\_removed += task.result;

}

}

cout << "\nSummary:" << endl;

cout << "Successfully processed files: " << (num\_tasks - failed\_tasks) << endl;

cout << "Failed files: " << failed\_tasks << endl;

cout << "Total characters removed: " << total\_removed << endl;

return failed\_tasks == 0 ? EXIT\_SUCCESS : EXIT\_FAILURE;

}







