Петров Андрей, 13 группа 3 курс

1. Создадим последовательную программу

```
static void sequential() {
    ifstream file("../fileS.txt");
    ofstream fileReverse("../fileS_reversed.txt");

    while (file.peek() != EOF) {
        string buffer;
        file >> buffer;
        reverse(buffer.begin(), buffer.end());
        fileReverse << buffer;
    }
    file.close();
    fileReverse.close();
}
```

2. Создадим produces/consumer;

```
static void producer() {
    ifstream file("../file.txt");
    while (file.peek() != EOF) {
        string buffer;
        file >> buffer;
        reverse(buffer.begin(), buffer.end());
        this_thread::sleep_for(10ms);
        {
            lock_guard<mutex> lk{mut};
            q.push(buffer);
        }
        cv.notify_all();
    }
    {
        lock_guard<mutex> lk{mut};
        finished = true;
    }
    cv.notify_all();
    file.close();
}

static void consumer() {
    ofstream file("../file_reversed.txt");
    while (!finished) {
        unique_lock<mutex> l{mut};
        cv.wait(l, [] { return !q.empty() || finished; });
        while (!q.empty()) {
            file << q.front();
            q.pop();
        }
    }
    file.close();
}
```

3. Замерим время

```
int main() {
    LARGE_INTEGER liFrequency, liStartTime, liFinishTime, liStartTime1,
liFinishTime1;
    double dElapsedTime;
    QueryPerformanceFrequency(&liFrequency);

    //=============================================
    QueryPerformanceCounter(&liStartTime);
    sequential();
    QueryPerformanceCounter(&liFinishTime);
    dElapsedTime = 1000.0 * (liFinishTime.QuadPart - liStartTime.QuadPart) /
                   liFrequency.QuadPart;
    printf("Sequential finished! Time:  %f\n", dElapsedTime);
    //=============================================

    //=============================================
    QueryPerformanceCounter(&liStartTime1);
    thread t1{producer};
    thread t2{consumer};
    t1.join();
    t2.join();
    QueryPerformanceCounter(&liFinishTime1);
    dElapsedTime = 1000.0 * (liFinishTime1.QuadPart - liStartTime1.QuadPart)
                   / liFrequency.QuadPart;
    printf("Producer/Consumer finished! Time:  %f\n", dElapsedTime);
    //=============================================
}
```

Размер файла, байты	Время выполнения	
	Последовательная программа	Produced/Consumer
1000	0.726800	2231.080000
10000	7.520700	23208.983800
100000	53.489900	230693.843100

ПОЛНЫЙ КОД ПРОГРАММЫ:
```cpp
#include <iostream>
#include <queue>
#include <condition_variable>
#include <thread>
#include <fstream>
#include <windows.h>
#include <cmath>

using namespace std;
using namespace chrono_literals;

queue<string> q;
mutex mut;
condition_variable cv;
bool finished{false};

static void sequential() {
    ifstream file("../fileS.txt");
    ofstream fileReverse("../fileS_reversed.txt");

    while (file.peek() != EOF) {
        string buffer;
        file >> buffer;
        reverse(buffer.begin(), buffer.end());
        fileReverse << buffer;
    }
    file.close();
    fileReverse.close();
}

static void producer() {
    ifstream file("../file.txt");
    while (file.peek() != EOF) {
        string buffer;
        file >> buffer;
        reverse(buffer.begin(), buffer.end());
        this_thread::sleep_for(10ms);
        {
            lock_guard<mutex> lk{mut};
            q.push(buffer);
        }
        cv.notify_all();
    }
    {
        lock_guard<mutex> lk{mut};
        finished = true;
    }
    cv.notify_all();
    file.close();
}

static void consumer() {
    ofstream file("../file_reversed.txt");
    while (!finished) {
        unique_lock<mutex> l{mut};
        cv.wait(l, [] { return !q.empty() || finished; });
        while (!q.empty()) {
            file << q.front();
            q.pop();
        }
    }
    file.close();
}


int main() {
    LARGE_INTEGER liFrequency, liStartTime, liFinishTime, liStartTime1,
liFinishTime1;
    double dElapsedTime;
    QueryPerformanceFrequency(&liFrequency);

    //============================================
    QueryPerformanceCounter(&liStartTime);
    sequential();
    QueryPerformanceCounter(&liFinishTime);
    dElapsedTime = 1000.0 * (liFinishTime.QuadPart - liStartTime.QuadPart) /
liFrequency.QuadPart;
    printf("Sequential finished! Time:  %f\n", dElapsedTime);
    //============================================

    //============================================
    QueryPerformanceCounter(&liStartTime1);
    thread t1{producer};
    thread t2{consumer};
    t1.join();
    t2.join();
    QueryPerformanceCounter(&liFinishTime1);
    dElapsedTime = 1000.0 * (liFinishTime1.QuadPart - liStartTime1.QuadPart)
/ liFrequency.QuadPart;
    printf("Producer/Consumer finished! Time:  %f\n", dElapsedTime);
    //============================================
}
```