Босько Виктор, 151004, Лабораторная работа №5

**Source.cpp**

#include <iostream>

#include <vector>

#include <queue>

#include <string>

#include <algorithm>

#include <fstream>

#include <Windows.h>

#include "AtomicQueue.h"

using namespace std;

// Data for sorting threads

struct ThreadData

{

AtomicQueue<string>\* q;

AtomicQueue<string>\* result;

};

// Initialization of the queue from file

static void getQ(string fileName, AtomicQueue<string>& q)

{

ifstream inputFile(fileName);

if (inputFile.is\_open())

{

string line;

while (getline(inputFile, line))

{

q.Enqueue(line);

}

inputFile.close();

}

else

{

cout << "File wasn't open" << endl;

}

}

// Sorting algorithm for threads

static DWORD WINAPI sortStrings(LPVOID lpParam)

{

ThreadData\* threadData = static\_cast<ThreadData\*>(lpParam);

string line;

while (!threadData->q->Dequeue(line))

{

sort(line.begin(), line.end());

threadData->result->Enqueue(line);

}

delete threadData;

return 0;

}

// Writing results to file and console

static void Output(string fileName, AtomicQueue<string> q)

{

cout << q.ToString();

ofstream outputFile(fileName);

if (outputFile.is\_open())

{

outputFile << q.ToString();

outputFile.close();

cout << "String successfully written to file" << endl;

}

else

{

cout << "Failed to open the file" << endl;

}

}

int main()

{

int numThreads;

string fileName;

AtomicQueue<string> q;

AtomicQueue<string> result;

cout << "Enter file path" << endl;

cin >> fileName;

cout << "Emter number of threads" << endl;

cin >> numThreads;

HANDLE\* threads = new HANDLE[numThreads];

ThreadData\*\* threadData = new ThreadData\* [numThreads];

getQ(fileName, q);

for (int i = 0; i < numThreads; i++)

{

threadData[i] = new ThreadData();

threadData[i]->q = &q;

threadData[i]->result = &result;

threads[i] = CreateThread(NULL, 0, sortStrings, threadData[i], 0, NULL);

}

WaitForMultipleObjects(numThreads, threads, TRUE, INFINITE);

for (int i = 0; i < numThreads; i++)

{

if (threads[i] != NULL)

{

CloseHandle(threads[i]);

}

}

Output(fileName, result);

delete[] threads;

delete[] threadData;

return 0;

}

**AtomicQueue.h**

#pragma once

#include <queue>

#include <string>

#include <Windows.h>

using namespace std;

// Thread safe queue with atomic operations

template<typename T>

class AtomicQueue

{

public:

AtomicQueue()

{

InitializeCriticalSection(&criticalSection);

}

~AtomicQueue()

{

DeleteCriticalSection(&criticalSection);

}

// Adds element to the queue

void Enqueue(const T& item)

{

EnterCriticalSection(&criticalSection);

q.push(item);

LeaveCriticalSection(&criticalSection);

}

// Removes element from queue if it is not empty to elem parameter, otherwise returns false

bool Dequeue(T& elem)

{

bool isEmpty = true;

EnterCriticalSection(&criticalSection);

if (!q.empty())

{

elem = q.front();

q.pop();

isEmpty = false;

}

LeaveCriticalSection(&criticalSection);

return isEmpty;

}

// Counts current size of the queue NOT THREAD SAFE TO USE

int Size()

{

return q.size();

}

// Checks if queue is empty NOT THREAD SAFE TO USE

bool IsEmpty()

{

return q.empty();

}

// Converts elements to string. Only for primitive types NOT THREAD SAFE

string ToString()

{

string res;

string line;

queue<T> q\_copy = q;

while (!q\_copy.empty())

{

res.append(q\_copy.front() + "\n");

q\_copy.pop();

}

return res;

}

private:

CRITICAL\_SECTION criticalSection;

queue<T> q;

};