**МИНОБРНАУКИ РОССИИ**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №6

Дисциплина: Объектно-ориентированное программирование

по теме Потоки.

Выполнил: ст. группы ВТ-22  
Макаров Даниил Сергеевич

Проверил: Буханов Д.Г.

**Белгород 2019**

***Лабораторная работа №6***

**Потоки в С++**

Цель работы: изучение основных возможностей потоков управления и потоков ввода-вывода. Получение навыков работы со стандартными средствами управления потоками в С++11. Знакомство с классом Thread и стандартными средствами синхронизации потоков.

Задание:

1. Изучить основные классы и их возможности работы с потоками в С++11.

2. Разработать программу в соответствии с вариантом задания. Программа должна содержать 2 объекта Thread, унаследованных от общего предка, в котором должны содержаться общие для обоих процессов свойства. Вывод организовать в отдельном потоке.

3. Разработать программу в соответствии с вариантом задания (номер варианта + 3), используя API CreateThread.

4. Сделать выводы о проделанной работе.

//Один поток генерирует входные данные для функции, F=(sin(x)+exp(2x))/tg(x) а другой поток вычисляет значение этой функции.

//Произвести синхронный вывод при каждой итерации. Показать выполнение работы программы в синхронном и асинхронном режимах.

#include <thread>

#include <mutex>

#include <iostream>

#include <math.h>

#include <time.h>

#include <unistd.h>

using namespace std;

class Thread{

double &data;

virtual void func() = 0;

virtual void func(mutex &m) = 0;

public:

Thread(double &data):data(data){};

double get\_data();

void set\_data(double new\_data);

void operator()();

void operator()(mutex &m);

};

class GenData:public Thread{

void func();

void func(mutex &m);

public:

GenData(double &data):Thread(data){

srand(time(NULL));

};

};

class ComputeData:public Thread{

void func();

void func(mutex &m);

double &result;

public:

ComputeData(double &data,double &result):Thread(data),result(result){};

double get\_result();

};

class OutputData:public Thread{

double &result;

void func();

void func(mutex &m);

public:

OutputData(double &data,double &result):Thread(data),result(result){};

};

double Thread::get\_data(){

return this->data;

};

void Thread::set\_data(double new\_data){

this->data=new\_data;

};

void Thread::operator()(){

return this->func();

};

void Thread::operator()(mutex &m){

return this->func(m);

};

void GenData::func(){

while(true){

set\_data((double)random());

}

};

void GenData::func(mutex &m){

while(true){

m.lock();

set\_data((double)random());

m.unlock();

}

};

void ComputeData::func(){

while(true){

result=(sin(get\_data())+exp(2\*get\_data()))/tg(get\_data());

}

};

void ComputeData::func(mutex &m){

while(true){

m.lock();

result=(sin(get\_data())+exp(2\*get\_data()))/tg(get\_data());

m.unlock();

}

};

void OutputData::func(){

while(true){

cout<<"X = "<<get\_data()<<"; f(X) = "<<result<<"; TRUE f(x) = "<<(sin(get\_data())+exp(2\*get\_data()))/tg(get\_data())<<endl;

usleep(1000000);

};

};

void OutputData::func(mutex &m){

while(true){

m.lock();

cout<<"X = "<<get\_data()<<"; f(X) = "<<result<<"; TRUE f(x) = "<<(sin(get\_data())+exp(2\*get\_data()))/tg(get\_data())<<endl;

m.unlock();

usleep(1000000);

};

};

int main(){

bool sync\_mode=false;

double data=0,result=0;

GenData thread\_1(data);

ComputeData thread\_2(data,result);

OutputData thread\_3(data,result);

cout<<"1 - sync, 0 - async"<<endl;

cin>>sync\_mode;

if(sync\_mode){

mutex m;

thread th\_gen(thread\_1,ref(m));

thread th\_calc(thread\_2,ref(m));

thread th\_out(thread\_3,ref(m));

th\_gen.detach();

th\_calc.detach();

th\_out.join();

}

else{

thread th\_gen(thread\_1);

thread th\_calc(thread\_2);

thread th\_out(thread\_3);

th\_gen.detach();

th\_calc.detach();

th\_out.join();

};

return 0;

}

//Один поток удаляет лишние пробелы в строке, а другой подсчитывает

//количество слов в тексте. Произвести синхронный вывод при каждой

//итерации. Показать выполнение работы программы в синхронном и

//асинхронном режимах.

#include <iostream>

#include <math.h>

#include <time.h>

#include <windows.h>

#include <string>

#include <cstdlib>

#include <ctime>

using namespace std;

class Thread {

private:

    CONST HANDLE hMutex;

    bool sync;

string &str;

public:

    friend void start\_thread(Thread&);

    virtual void func() = 0;

    Thread(CONST HANDLE hMutex, bool sync, std::string& ) : hMutex(hMutex), sync(sync), str(str) {};

    void lock\_mutex() { WaitForSingleObject(this->hMutex, INFINITE); }

    void unlock\_mutex() { ReleaseMutex(this->hMutex); }

    bool get\_sync() { return this->sync; }

string &get\_str() { return this->str; }

};

DWORD WINAPI start\_thread(CONST LPVOID lpParam) {

    ((Thread\*)lpParam)->func();

    ExitThread(0);

}

class WordCounter : public Thread {

unsigned &count\_of\_word;

public:

    void func();

    WordCounter(CONST HANDLE hMutex, bool sync,unsigned &count\_of\_word, std::string& str) : Thread(hMutex, sync, str),count\_of\_word(count\_of\_word) {};

};

class SpaceEraser : public Thread {

public:

    void func();

SpaceEraser(CONST HANDLE hMutex, bool sync, std::string& str) : Thread(hMutex, sync, str) {};

};

class OutThread : public Thread {

unsigned &count\_of\_word;

public:

    void func();

    OutThread(CONST HANDLE hMutex, bool sync,unsigned &count\_of\_word, std::string& str) : Thread(hMutex, sync, str),count\_of\_word(count\_of\_word) {};

};

void SpaceEraser::func(){

for(unsigned i=1;i<get\_str().size();i++){

if(get\_sync()) lock\_mutex();

if(get\_str()[i]==' '){

if(get\_str()[i-1]==' '){

get\_str().erase(i,1);

};

};

};

};

int get\_word(string &str,int &current\_char){

int char\_in\_word=0;

while((current\_char<str.size())&&(str[current\_char]!=' ')){

current\_char++;

char\_in\_word++;

}

return char\_in\_word;

};

void WordCounter::func(){

while(true){

if(get\_sync()) lock\_mutex();

count\_of\_word=0;

int current\_char=0;

while(current\_char<get\_str().size()){

if(get\_word(get\_str(),current\_char)>0) count\_of\_word++;

current\_char++;

};

if(get\_sync()) unlock\_mutex();

};

};

void OutThread::func(){

while(true){

this->lock\_mutex();

cout<<endl<<get\_str()<<endl<<"Count of word - "<<count\_of\_word<<endl;

this->unlock\_mutex();

Sleep(300);

}

}

int main(){

string s;

bool sync\_mode=true;

unsigned count\_of\_word;

cout<<"Input str: ";

getline(cin,s);

cout<<"1 - sync, 0 - async:"<<endl;

cin>>sync\_mode;

CONST HANDLE Mutex = CreateMutex(NULL, FALSE, NULL);

    HANDLE hThreads[3];

WordCounter th1(Mutex,sync\_mode,ref(count\_of\_word),ref(s));

SpaceEraser th2(Mutex,sync\_mode,ref(s));

OutThread th3(Mutex,sync\_mode,ref(count\_of\_word),ref(s));

hThreads[0] = CreateThread(NULL, 0, &start\_thread, &th1, 0, NULL);

    hThreads[1] = CreateThread(NULL, 0, &start\_thread, &th2, 0, NULL);

    hThreads[2] = CreateThread(NULL, 0, &start\_thread, &th3, 0, NULL);

    WaitForMultipleObjects(3, hThreads, TRUE, INFINITE);

CloseHandle(hThreads[0]);

    CloseHandle(hThreads[1]);

    CloseHandle(hThreads[2]);

    CloseHandle(Mutex);

    ExitProcess(0);

return 0;

}