**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

Микропроект №2

Вариант 3

**Исполнитель**

Студент группы БПИ195/1

Пучин М.А.

Задание

3. Задача о читателях и писателях. Базу данных разделяют два типа процессов – читатели и писатели. Читатели выполняют транзакции, которые просматривают записи базы данных, транзакции писателей и просматривают и изменяют записи. Предполагается, что в начале БД находится в 4 непротиворечивом состоянии (т.е. отношения между данными имеют смысл). Каждая отдельная транзакция переводит БД из одного непротиворечивого состояния в другое. Для предотвращения взаимного влияния транзакций процесс-писатель должен иметь исключительный доступ к БД. Если к БД не обращается ни один из процессов-писателей, то выполнять транзакции могут одновременно сколько угодно читателей. Создать многопоточное приложение с потоками-писателями и потоками-читателями. Реализовать решение, используя семафоры.

Состав​​ программы

Входные данные представлены в виде двух чисел: количеств потоков писателей и читателей.

В программе реализованы классы:

1. Note – класс-запись в БД. Содержит 2 поля: имя автора и текст записи.

Метод GetText позволяет получить текст записи.

Метод GetWriterID возращает id писателя.

Метод SetText записывает новый текст.

1. Writer – класс-писатель. Задается ID (номер потока в дальнейшем). Может читать и писать, а также имеет метод, возвращающий ID объекта.
2. Reader – класс-читатель. Умеет читать и задается также ID читателя (номер потока в дальнейшем).
3. Также в программе присутствуют ключевые функции CreateWriters (заполняет массив писателей объектами класса), CreateReaders (заполняет массив читателей), FillNotes (заполняет БД записями (по 5 от каждого писателя, изначально в них текст “Empty”), TimeTest – функция, возвращающая строку, представляющую текущее время, StartWriter – функция, симулирующая действие писателя и StartReader – симулятор читателя.

В main создаются потоки StartWriter и StartReader.

Рассмотрим поподробнее функции-симуляторы.

Обе функции содержат бесконечный цикл.

В читателе: поток ждет пока семафор == true, затем проходит по всем записям.

В писателе: функция синхронизирована, только один писатель может менять записи одновременно. Функция проходит по элементам БД, находит записи со своим ID и меняет их на “OpenMP”. Затем уведомляет остальные потоки о возможности читать или писать.

Текст программы:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <vector>

#include <thread>

#include <mutex>

#include <cstdlib>

#include <ctime>

#include <iomanip>

#include <chrono>

//Вариант 3

using namespace std;

using namespace chrono;

int amount\_of\_writers = 0;

int amount\_of\_readers = 0;

int iter = 10;

bool sem\_writer = false;

condition\_variable cv;

mutex mtx;

string gen\_string() {

return "OpenMP";

}

class Note {

private:

string text;

int writer\_id;

public:

Note(string text\_, int writer\_id\_) {

text = text\_;

writer\_id = writer\_id\_;

}

string GetText() {

return text;

}

int GetWriterID() {

return writer\_id;

}

void SetText(string text\_) {

text = text\_;

return;

}

};

class Reader {

private:

int reader\_id;

public:

Reader() {

reader\_id = amount\_of\_readers++;

}

int GetID() {

return reader\_id;

}

void Read(vector<Note>& notes, int note\_id) {

//reading

string note = notes[note\_id].GetText();

return;

}

};

class Writer {

private:

int writer\_id;

public:

Writer() {

writer\_id = amount\_of\_writers++;

}

void Write(vector<Note>& notes, int note\_id) {

//writing

Note n = notes[note\_id];

n.SetText(gen\_string());

notes[note\_id] = n;

return;

}

int GetID() {

return writer\_id;

}

void Read(vector<Note>& notes, int note\_id) {

//reading

string note = notes[note\_id].GetText();

return;

}

};

void CreateWriters(int amount, vector<Writer>& writers) {

for (int i = 0; i < amount; i++) {

writers.push\_back(Writer());

}

return;

}

void CreateReaders(int amount, vector<Reader>& readers) {

for (int i = 0; i < amount; i++) {

readers.push\_back(Reader());

}

return;

}

void FillNotes(vector<Note>& notes, vector<Writer>& writers) {

for (auto writer : writers) {

int id = writer.GetID();

for (int i = 0; i < 5; i++) {

notes.push\_back(Note("Empty", id));

}

}

return;

}

string TimeTest()

{

time\_t seconds = time(NULL);

tm\* timeinfo = localtime(&seconds);

return asctime(timeinfo);

}

void StartWriter(vector<Note> &notes, Writer writer) {

while (true) {

if (iter == 0) {

cout << "Last iteration done. Exiting...";

exit(0);

}

unique\_lock<mutex> ulck(mtx);

iter--;

sem\_writer = true;

string time1 = TimeTest();

cout << time1.substr(0, time1.size() - 1) << " \*\*\*\* Writer with id: " << writer.GetID() << " started changing his notes..." << endl;

for (int i = 0; i < notes.size(); i++) {

if (notes[i].GetWriterID() == writer.GetID() && notes[i].GetText() == "Empty") {

writer.Write(notes, i);

break;

}

}

time1 = TimeTest();

cout << time1.substr(0, time1.size() - 1) << " \*\*\*\* Writer with id: " << writer.GetID() << "; Now notes look like: " << endl;

for (int i = 0; i < notes.size(); i++) {

cout << "[ #" << i << ": " << "Author: " << notes[i].GetWriterID() << "; Text: " << notes[i].GetText() << " ]" << endl;

}

time1 = TimeTest();

cout << time1.substr(0, time1.size() - 1) << " \*\*\*\* Writer with id: " << writer.GetID() << " finished changing his notes... Now he is afk for 1 second." << endl;

sem\_writer = false;

ulck.unlock();

cv.notify\_all();

std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

}

}

void StartReader(vector<Note>& notes, Reader reader) {

while (true) {

unique\_lock<mutex> lck(mtx);

while (sem\_writer == true) {

cv.wait(lck);

}

string time1 = TimeTest();

cout << time1.substr(0, time1.size() - 1) << " ---- Reader with id: " << reader.GetID() << " started searching the notes..." << endl;

lck.unlock();

for (int i = 0; i < notes.size(); i++) {

reader.Read(notes, i);

}

lck.lock();

time1 = TimeTest();

cout << time1.substr(0, time1.size() - 1) << " ---- Reader with id: " << reader.GetID() << " finished searching the notes. Now he is afk for 1 sec. " << endl;

lck.unlock();

std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

}

}

signed main() {

int n, k;

cout << "Enter amount of writers: ";

cin >> n;

while (true)

{

if (cin.fail())

{

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Wrong input! Try again: ";

cin >> n;

}

if (!cin.fail())

break;

}

cout << "Enter amount of readers: ";

cin >> k;

while (true)

{

if (cin.fail())

{

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Wrong input! Try again: ";

cin >> k;

}

if (!cin.fail())

break;

}

vector<Note> notes;

vector<Writer> writers;

vector<Reader> readers;

CreateWriters(n, writers);

CreateReaders(k, readers);

FillNotes(notes, writers);

for (auto x : notes) {

cout << x.GetWriterID() << x.GetText() << endl;

}

thread \*vec\_reader = new thread[k];

thread\* vec\_writer = new thread[n];

for (int reader\_index = 0; reader\_index < readers.size(); reader\_index++) {

vec\_reader[reader\_index] = thread(StartReader, ref(notes), readers[reader\_index]);

}

for (int writer\_index = 0; writer\_index < writers.size(); writer\_index++) {

vec\_writer[writer\_index] = thread(StartWriter, ref(notes), writers[writer\_index]);

}

for (int i = 0; i < k; i++) {

vec\_reader[i].join();

}

for (int i = 0; i < n; i++) {

vec\_writer[i].join();

}

}