**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

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

Факультет компьютерных наук

Департамент программной инженерии

Многопоточное программирование.

Взаимодействие потоков. Микропроект. Вариант 3

**Исполнитель**

Студент группы БПИ191

Архаров Д. П.

Задание

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

Решение:

Программа использует в качестве базы данных одномерный массив. Потоки читатели считывают элемент массива, потоки писатели записывают элементы. Во время того как с базой работает поток писатрель ему предоставляется исключительный доступ(unique\_lock), предотвращающий конфликты транзакций. Во время же работы потока читателя устанавливается совместный доступ(shared\_lock). Таким образом несколько читателей могут читать базу данных одновременно, если нет писателя, а записывать в базу может только один поток одновременно. Обработка доступа осуществлена с помощью shared\_mutex осуществляющего функционал ReadWriteLock-а, одного из простейших конструктов для синхронизации потоков. Так как это mutex, то есть двоичный семафор, работа выполнена с использованием семафоров.

Используемые Источники :

<http://softcraft.ru/edu/comparch/> - Сайт учебной программы

<https://en.cppreference.com/w/cpp/thread/shared_mutex> - Документация по работе с Shared\_mutex

Текст программы:

// Архаров Дмитрий Павлович

// БПИ191

// 3-ий Вариант.

#include <stdlib.h>

#include <stdio.h>

#include <thread>

#include <shared\_mutex>

#include <ctime>

#include <iostream>

#include <vector>

#include <chrono>

using namespace std;

const int N = 10; // Размер базы данных

int Data[N]; // Используемая база данных (одномерный массив)

shared\_mutex RWlock; // Используем shared\_mutex как реализацию ReadWriteLock, кроме того мьютекс это по сути семафор.

void Reader(int num) // Функция для потоков читателей

{

while (true)

{

shared\_lock<shared\_mutex> readlock(RWlock); // Устанавливаем shared\_lock то есть совместное использование,позволяющее просматривать данные нескольким потокам одновременно

int x = rand() % N; // Просматриваем случайную ячейку из базы данных

int a = Data[x]; // Считываем данные из ячейки

readlock.unlock(); // Снимаем наш shared\_lock

printf("Время %d | Читатель %d: Элемент[%d] Получен -> %d\n", clock(),num, x, a); // Выводим в консоль информацию о просмотренном вместе с временем

}

}

void Writer(int num) // Функция для потоков писателей

{

while (true)

{

unique\_lock<shared\_mutex> writelock(RWlock); // Устанавливаем unique\_lock передающий исключительные полномочия нашему потоку писателю

int x = rand() % N; // Выбираем случайную ячейку

Data[x] = rand() % 10 + 1 + num; // Меняем её значение

writelock.unlock(); // Снимаем unique\_lock

fprintf(stdout, "Время %d | Писатель %d: Элемент[%d] = % d\n", clock(), num, x, Data[x]); // Выводим в консоль информацию об изменении вместе с временем

}

}

int main()

{

setlocale(LC\_ALL, "Russian"); // This project uses russian language

// Инциалиизация начальных данных :

int numRead = -1, numWrite = -1;

while (numRead < 0) // Не даём ввести отрицательное кол-во потоков, поскольку иначе всё сломается

{

cout << "Введи кол-во читателей (Больше или равно нуля) :";

cin >> numRead;

}

while (numWrite < 0)

{

cout << "Введи кол-во писателей (Больше или равно нуля) :";

cin >> numWrite;

}

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

{

Data[i] = 1;

}

// Инциализация потоков :

vector <thread> Readers(numRead);

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

{

Readers[i] = thread(Reader, i+1);

}

vector <thread> Writers(numWrite);

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

{

Writers[i] = thread(Writer, i+1);

}

// Join потоков :

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

{

Readers[i].join();

}

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

{

Writers[i].join();

}

return 0;

}