

Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Разработка многопоточного приложения

Отчет по лабораторной работе №7 дисциплины
«Технологии программирования»

Выполнил студент группы ИВТ-22_____ /Крючков И. С/
Проверил _____ /Долженкова М. Л./

Киров 2022

1. Задание

Написать многопоточное приложение, решающее задачу о читателях и писателях. При реализации использовать Mutex.

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 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 одно временно сколько угодно читателей.

2. Теория

Мьютекс – объект ядра, который создается с помощью функции CreateMutex. С помощью него хорошо защищать ресурс от одновременного обращения к нему разными потоками. В один момент времени только один поток владеет мьютексом. Мьютекс может быть известен одновременно нескольким потокам. При вызове функции CreateMutex только первый поток создает мьютекс а все остальные получают идентификатор уже существующего объекта. Это дает возможность нескольким процессам управлять одним и тем же мьютексом избавляя программиста от необходимости отслеживать от освобождения памяти под объект. В противном случае возникают трудности с созданием мьютекса. Если мьютекс создан в родительском потоке то его идентификатор является глобальным и доступен для всех нижесозданных потоков. Освобождение мьютекса осуществляется с помощью ReleaseMutex после чего мьютекс может быть захвачен другим потоком.

3. Результаты работы программы

```
13:26:51:608 START
13:26:51:609 READ (1)
13:26:51:610 READ (2)
13:26:51:610 READ (3)
13:26:51:610 WRITE
13:26:51:610 WRITE
13:26:53:632 READ (3)
13:26:53:632 READ (1)
13:26:53:632 READ (2)
13:26:55:626 WRITE
13:26:55:626 WRITE
13:26:55:657 READ (3)
13:26:55:657 READ (2)
13:26:55:657 READ (1)
13:26:57:680 READ (3)
13:26:57:680 READ (1)
13:26:57:680 READ (2)
13:26:59:643 WRITE
13:26:59:643 WRITE
13:26:59:704 READ (2)
13:26:59:704 READ (3)
13:26:59:704 READ (1)
13:27:1:721 READ (3)
13:27:1:721 READ (2)
13:27:1:721 READ (1)
13:27:3:665 WRITE
13:27:3:665 WRITE
13:27:3:742 READ (1)
13:27:3:742 READ (2)
13:27:3:742 READ (3)
13:27:5:760 READ (2)
13:27:5:760 READ (3)
13:27:5:760 READ (1)
13:27:7:688 WRITE
13:27:7:688 WRITE
13:27:7:783 READ (3)
13:27:7:783 READ (2)
13:27:7:783 READ (1)
```

Рисунок 1 – Результат работы программы с использованием потоков

```

13:35:43:812 START
13:35:43:812 READ (1)
13:35:45:844 READ (1)
13:35:47:861 READ (1)
13:35:49:887 READ (1)
13:35:51:918 READ (1)
13:35:53:941 READ (1)
13:35:55:962 READ (1)
13:35:57:981 READ (1)
13:36:0:6 READ (1)
13:36:2:37 READ (1)
13:36:4:61 READ (2)
13:36:6:92 READ (2)
13:36:8:123 READ (2)
13:36:10:155 READ (2)
13:36:12:186 READ (2)
13:36:14:217 READ (2)
13:36:16:248 READ (2)
13:36:18:280 READ (2)
13:36:20:311 READ (2)
13:36:22:342 READ (2)
13:36:24:361 READ (3)
13:36:26:387 READ (3)
13:36:28:394 READ (3)
13:36:30:425 READ (3)
13:36:32:457 READ (3)
13:36:34:488 READ (3)
13:36:36:519 READ (3)
13:36:38:550 READ (3)
13:36:40:582 READ (3)
13:36:42:613 READ (3)
13:36:44:644 WRITE
13:36:48:675 WRITE
13:36:52:707 WRITE
13:36:56:720 WRITE
13:37:0:750 WRITE
13:37:4:781 WRITE
13:37:8:812 WRITE
13:37:12:844 WRITE

```

Рисунок 2 – Результат работы программы без использования потоков

4. Листинг кода

```

#include <iostream>
#include <thread>
#include <mutex>
#include <ctime>
#include <windows.h>

int iterNum = 10;
std::mutex mx_io;
int count = 0;

HANDLE hmx = CreateMutex(NULL, false, NULL);

bool ready = true;
struct mTime {
    int h;
    int m;
    int s;
    int ms;

```

```

};

mTime getTime(){
    mTime nt;
    SYSTEMTIME st;

    GetLocalTime(&st);

    nt.h = st.wHour;
    nt.m = st.wMinute;
    nt.s = st.wSecond;
    nt.ms = st.wMilliseconds;
    return nt;
}

void writer(){
    WaitForSingleObject(hmx, 0);

    mTime ct = getTime();
    {
        std::lock_guard<std::mutex> io_lock(mx_io);
        std::cout << ct.h << ":" << ct.m << ":" << ct.s << ":" << ct.ms << " WRITE" << std::endl;
    }

    std::this_thread::sleep_for(std::chrono::seconds(1));

    ReleaseMutex(hmx);
}

void reader(int id){
    WaitForSingleObject(hmx, 0);
    ReleaseMutex(hmx);

    mTime ct = getTime();

    {
        std::lock_guard<std::mutex> io_lock(mx_io);
        std::cout << ct.h << ":" << ct.m << ":" << ct.s << ":" << ct.ms << " READ (" << id << ")"
<< std::endl;
    }

    std::this_thread::sleep_for(std::chrono::seconds(1));
}

void run(int id, int dl){
    for(int i = 0; i < iterNum; i++){

        reader(id);
        std::this_thread::sleep_for(std::chrono::seconds(dl));
    }
}

```

```

void runWriter(){
    for(int i = 0; i < iterNum; i++){

        writer();
        std::this_thread::sleep_for(std::chrono::seconds(3));
    }
}

int main(){
    auto start_time = std::chrono::steady_clock::now();
    mTime ct = getTime();
    std::cout << ct.h << ":" << ct.m << ":" << ct.s << ":" << ct.ms << " START" << std::endl;

    std::thread run1(run, 1, 1);
    std::thread run2(run, 2, 1);
    std::thread run4(run, 3, 1);
    std::thread run3(runWriter);
    std::thread run5(runWriter);
    run1.join();
    run2.join();
    run3.join();
    run4.join();
    run5.join();
    auto end_time = std::chrono::steady_clock::now();
    auto elapsed_ns = std::chrono::duration_cast<std::chrono::milliseconds>(end_time - start_time);
    std::cout << "COMPLETE " << elapsed_ns.count() << " ms\n";

    return 1;
}

```

5. Вывод

В ходе выполнения лабораторной работы балы написана многопоточная программа, решающая задачу о читателя и писателях. Для синхронизации поток использованы мьютексы. Также было подсчитано время работы программы в однопоточном и многопоточном режимах, в результате, при использовании потоков производительность выросла в 4 раза.