МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчёт по лабораторной работе №4

по дисциплине «Параллельное программирование»

**Синхронизация потоков**

Выполнил: студент группы ФИб-3301-51-00

/ А. В. Чурин/

/ В.А. Бызов /

Проверил: к.ф.-м.н. доцент каф. ПМиИ

Киров 2022

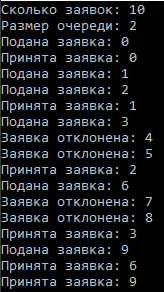
**Цель лабораторной работы**

Изучить классические задачи синхронизации. Обеспечить правильную работу программы с использованием различных примитивов синхронизации.

**Задание на лабораторную работу**

1. «Производители – потребители». Два потока обрабатывают заявки. Первый поток читает заявку и помещает её в очередь размера N, если очередь переполнена, то заявке отказывается. Второй поток берет заявки из очереди и обрабатывает их, если в очереди заявок нет, то поток засыпает. Написать параллельную программу, реализующую правильную работу этих потоков.

**Результат работы:**

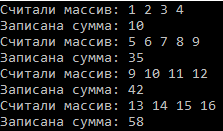
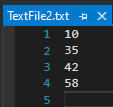
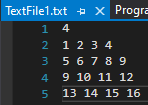


**Выводы:**

Для реализации данной задачи потребовался мьютекс, чтобы не было ошибок, связанных с одновременным использованием очереди разными потоками.

1. «Фоновая обработка». Два потока обрабатывают файл, содержащий набор целочисленных массивов. В файле находится M массивов, каждый в отдельной строке. В первой строке указано число M – количество массивов, а в следующих M строках записаны элементы массивов, разделенные пробелами. Первый поток по очереди читает массивы из файла, а второй по очереди считает суммы элементов этих массивов. Результаты первый поток должен записать в новый файл. Каждую сумму – в отдельной строке. Организовать правильную работу потоков.

**Результат работы:**

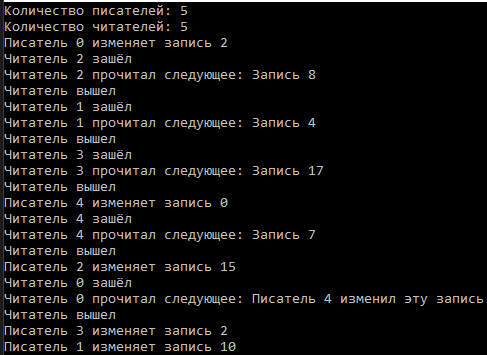


**Выводы:**

Для реализации потребовались очередь, в которую заносились строки чисел для суммирования, и один мьютекс, для контролирования доступа к очереди. Один поток сначала считывает строки и заносит их в очередь, а второй параллельно берёт эти строки, считает сумму и заносит в другую очередь. Доступ к очередям контролируется мьютексом.

1. «Читатели – писатели». Имеется хранилище данных, с которым работают одновременно несколько потоков. Первые N потоков случайным образом изменяют 2 данные в хранилище (при этом изменять данные в некоторый момент времени может ровно один поток), остальные M потоков периодически читают данные из хранилища. Несколько читателей могут находиться в хранилище одновременно, но читатель и писатель не могут одновременно использовать хранилище. Эмулировать работу хранилища.

**Результат работы:**



**Выводы:**

Для реализации потребовалось использовать семафоры. Один для того, чтобы давать доступ к хранилищу (на чтение и на запись), а другой следил, есть ли читатели в хранилище, так как у писателей мог быть доступ только тогда, когда читателей нет в хранилище.

**Выводы по лабораторной работе:**

По результатам данной лабораторной работы я изучил классические задачи синхронизации. Обеспечил правильную работу программы с использованием различных примитивов синхронизации.

**Листинг программы**

**Задание 1**

using System;

using System.Collections.Generic;

using System.Threading;

namespace Lab4

{

class Program

{

static Queue<String> queue = new Queue<string>();

static bool isEnd = false;

static Mutex mutexObj = new Mutex();

static int sizeQueue = 3;

static void Main(string[] args)

{

Console.Write("Сколько заявок: ");

int countRequest = int.Parse(Console.ReadLine());

Console.Write("Размер очереди: ");

sizeQueue = int.Parse(Console.ReadLine());

Thread manufacturer = new Thread(pushRequest);

Thread consumer = new Thread(readRequest);

manufacturer.Start(countRequest);

consumer.Start();

}

private static void readRequest()

{

while (!isEnd || queue.Count != 0)

{

if (queue.Count != 0)

{

mutexObj.WaitOne();

Console.WriteLine("Принята заявка: " + queue.Dequeue());

mutexObj.ReleaseMutex();

Thread.Sleep(1500);

}

else

{

Thread.Sleep(500);

}

}

}

private static void pushRequest(object countRequest)

{

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

{

Thread.Sleep(500);

if (queue.Count < sizeQueue)

{

mutexObj.WaitOne();

queue.Enqueue(i.ToString());

Console.WriteLine("Подана заявка: " + i.ToString());

mutexObj.ReleaseMutex();

}

else

{

Console.WriteLine("Заявка отклонена: " + i.ToString());

}

}

isEnd = true;

}

}

}

**Задание 2**

using System;

using System.Collections.Generic;

using System.IO;

using System.Threading;

namespace Lab4\_2

{

class Program

{

static Queue<List<int>> queue = new Queue<List<int>>();

static Queue<int> queueSum = new Queue<int>();

static bool isEndRead = false;

static bool isEndWrite = false;

static Mutex mutexObj = new Mutex();

static void Main(string[] args)

{

Thread thread1 = new Thread(readFromFile);

Thread thread2 = new Thread(sumMassive);

thread1.Start();

thread2.Start();

}

private static void readFromFile()

{

string path = "../../../TextFile1.txt";

using (StreamReader reader = new StreamReader(path))

{

int size = int.Parse(reader.ReadLine());

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

{

string[] array = reader.ReadLine().ToString().Split(" ");

List<int> massive = new List<int>();

for (int k = 0; k < array.Length; k++)

{

massive.Add(int.Parse(array[k]));

}

mutexObj.WaitOne();

queue.Enqueue(massive);

mutexObj.ReleaseMutex();

}

isEndRead = true;

}

path = "../../../TextFile2.txt";

using (StreamWriter writer = new StreamWriter(path))

{

while (!isEndWrite || queueSum.Count != 0)

{

if (queueSum.Count != 0)

{

int sum = queueSum.Dequeue();

writer.WriteLine(sum);

Console.WriteLine($@"Записана сумма: {sum}");

}

else

{

Thread.Sleep(500);

}

}

}

}

private static void sumMassive()

{

while (!isEndRead || queue.Count != 0)

{

if (queue.Count != 0)

{

mutexObj.WaitOne();

List<int> massive = queue.Dequeue();

mutexObj.ReleaseMutex();

string strMassive = "";

massive.ForEach(num => strMassive += num.ToString() + " ");

Console.WriteLine("Считали массив: " + strMassive);

int sum = 0;

massive.ForEach(num => sum += num);

queueSum.Enqueue(sum);

Thread.Sleep(1500);

}

else

{

Thread.Sleep(500);

}

}

isEndWrite = true;

}

}

}

**Задание 3**

using System;

using System.Collections.Generic;

using System.IO;

using System.Threading;

namespace Lab4\_3\_1

{

class Program

{

static List<String> storage = new List<String>();

static Mutex mutexObj = new Mutex();

static Semaphore ReadSem = new Semaphore(1, 1);

static Semaphore Access = new Semaphore(1, 1);

static int ReadCount = 0;

static void Main(string[] args)

{

Console.Write("Количество писателей: ");

int writerCount = int.Parse(Console.ReadLine());

List<Thread> writers = new List<Thread>();

Console.Write("Количество читателей: ");

int readerCount = int.Parse(Console.ReadLine());

List<Thread> readers = new List<Thread>();

for (int i = 0; i < 20; i++)

{

storage.Add($@"Запись {i}");

}

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

{

writers.Add(new Thread(changeStorage));

}

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

{

readers.Add(new Thread(readStorage));

}

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

{

writers[i].Start(i);

}

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

{

readers[i].Start(i);

}

}

private static void changeStorage(object idWriter)

{

Random random = new Random();

Thread.Sleep(random.Next(1000, 5000));

Access.WaitOne();

int idNote = random.Next(storage.Count - 1);

Console.WriteLine($@"Писатель {(int)(idWriter)} изменяет запись {idNote}");

storage[idNote] = $@"Писатель {(int)(idWriter)} изменил эту запись";

Access.Release();

}

private static void readStorage(object idReader)

{

Random random = new Random();

Thread.Sleep(random.Next(1000, 5000));

ReadSem.WaitOne();

ReadCount++;

if (ReadCount == 1)

{

Access.WaitOne();

}

ReadSem.Release();

Console.WriteLine("Читатель " + (int)(idReader) + " зашёл");

String note = storage[random.Next(storage.Count - 1)];

Console.WriteLine($@"Читатель {(int)(idReader)} прочитал следующее: {note}");

ReadSem.WaitOne();

ReadCount--;

if(ReadCount == 0)

{

Access.Release();

}

Console.WriteLine("Читатель вышел");

ReadSem.Release();

}

}

}