Оглавление

[Введение 1](#_Toc53586126)

[Ситуация гонки 1](#_Toc53586127)

[Автономная работа 2](#_Toc53586128)

[Минимальный программный код 3](#_Toc53586129)

# Введение

Первая проблема, встречающаяся при выполнении метода IncrementCount, -- это ситуация, когда осуществляется попытка одновременного изменения записи с опредленным идентификатором id. На рисунке 1 показано, к чему может привести ситуация гонки нескольких потоков. В записи БД с id = 5 в свойстве Counter хранилось значение 12. Вызвав метод IncrementCount(“5”,-4), операция UpSert была выполненна некооректно из-за вмешательства другого потока. 12-5!=4. Примечание: в данной задаче потоки моделируют одновременное обращение несколько пользователей БД к записи.

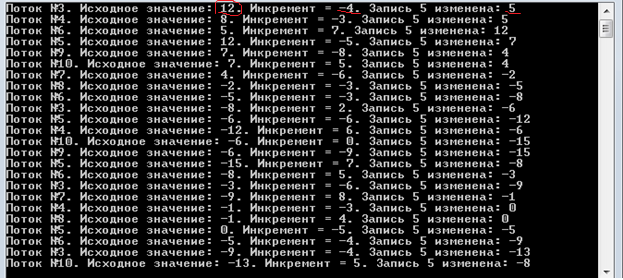


Рисунок 1 - Пример результата работы программы в состоянии гонки

# Ситуация гонки

Ситуация гонки может возникнуть по двум причинам из-за:

А) Одновременного обращения из текущей копии запущенной программы. Например, какой-то фоновый поток начал выполнять операцию IncrementCount и в этот же момент пользователь нажимает на кнопку интерфейса программы. Обработчик кноки также вызывает метод IncrementCount. Здесь может возникнуть ситуация одновременого создания/изменения записи.

Решение: использовать инструменты синхронизации доступа к критической секции, а именно конструкцию lock (object) {}. В данном случае критической секцией является тело метода IncrementCount. Коректная работа программы после применения синхронизации потоков отражена на рисунке 2. К слову, это приводит к некоторому увелечению времени выполнения программы, так как потокам приходится ждать своей очереди на получие доступа к критической секции.

Б) Одновременное обращения к одной и той же записи в БД из разных копий программ-клиентов. В данном случае возникновение исключения предотвратить невозможно, но можно реализовать его обработку. В теле метода IncrementCount осуществляется обработка исключения типа DuplicateKeyException и всех остальных исключений.

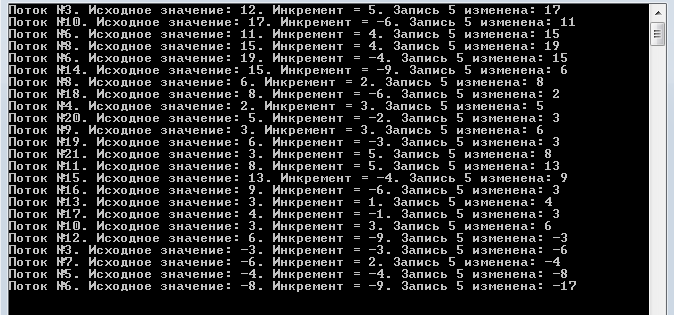


Рисунок 2 - Корректная работа программы после введения синхронизации потоков

# Автономная работа

В задании предявляется следующее требование: «…стоит отметить, что нам очень важно, сохранить изменения, даже если периодически возникают какие-то сбои, нам необходимо их каким-то образом попытаться обойти.»

Для реализации этого требование можно сделать следующее:

А) Попытаться выполнить операцию через некоторое время. В программе для этого есть две глобальные переменные: countOfAttemp – количество попыток выполнения операции, pauseBetweenAttemp – время ожидания между попытками в милисекундах. По умолчанию, значения переменных проиинициализированны занчениями 3 и 0 соотвественно.

Б) Сохранять все невыполненные операции в очередь, включая аргументы для этих операций. После возобновления соединения с сервером выполнить все операции, которые размещены в очереди. Или пытаться их выполнить через относительно недолгий промежуток времени 5-10 минут. Пример работы программы на рисунке 3. Операции, хранящиеся в очереди, выполняются после фразы «Остаток:».

В) Если подключение к серверу недоступно продолжительное время, то следует сохранить все невыполненные операции в файл. А после возобновления соединения считать файл и довыполнить операции. В отлиичии от решения Б, в котором очередь существует только тогда, когда существует процесс, в данном решении B, список операции хранится в файле, что позволяет перезагружать программу, устройство.

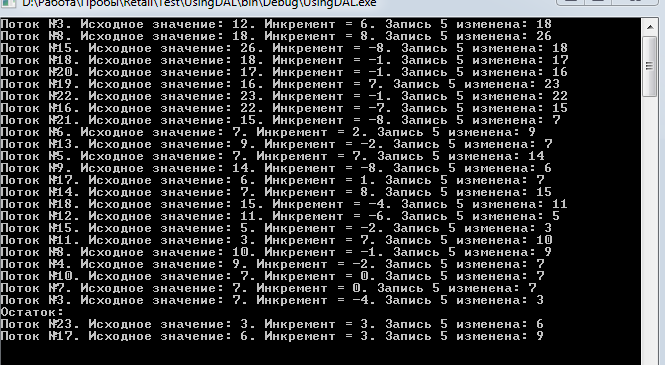


Рисунок - Использование очереди для довыполнения операций

Время, затраченное на выполнение задания (ПО + отчет): 2 ч.

# Минимальный программный код

int countOfAttemp = 3; // количество попыток выполнить операцию Upsert

int pauseBetweenAttemp = 0;//время ожидания между попытками

//объявление класса-делегата для хранения ссылки на IncrementCount

delegate void IncrementCountDelegate(string id, int increment);

//объект синхронизации доступа. Используется в конструкции lock. Им может быть любой экземпляр ссылочного типа.

string objectForSync = "Объект синхронизации доступа";

//используется для сохранения невыполненных операций типа IncrementCountDelegate и аргументов, поэтому закрыт параметром-типом object

Queue<object> restOfJobs = new Queue<object>();

public void IncrementCount(string id, int increment)

{

//глоабальная переменная countOfAttemp задает количество попыток выполнения операции Upsert .

for (int i = 0; i < this.countOfAttemp; i++)

{

//начало критической секции

lock (objectForSync)

{

try

{

this.mapper.Update(id: id, update: record => record.Counter + increment, upsert: true);

break; //выход из цикла в случаи успешного вполнения операции

}

catch (DuplicateKeyException ex)

{

//обработка исключения DuplicateKeyException

Console.WriteLine("Запись "+id+" уже существует." + ex.Message);

if (i == this.countOfAttemp - 1)

{

//запоминание невыполненной операции в стек или файл (нереализовано)

//После возобнолвения работоспособности сервера

//можно считать стек/файл и выполнить все невыполненные операции, сохраненные во время автономной работы

restOfJobs.Enqueue(new IncrementCountDelegate(IncrementCount));

restOfJobs.Enqueue(id);

restOfJobs.Enqueue(countOfAttemp);

}

}

catch (Exception ex)

{

//обработка исключения

Console.WriteLine("Сервер упал или уборщица выдернула шнур." + ex.Message);

if (i == this.countOfAttemp - 1)

{

//запоминание невыполненной операции в стек или файл (нереализовано)

//После возобнолвения работоспособности сервера

//можно считать стек/файл и выполнить все невыполненные операции, сохраненные во время автономной работы

restOfJobs.Enqueue(new IncrementCountDelegate(IncrementCount));

restOfJobs.Enqueue(id);

restOfJobs.Enqueue(countOfAttemp);

}

}

}

Thread.Sleep(pauseBetweenAttemp); //пауза между попытка выполнения операции Upsert

}

}

public void DoRestJobs()

{ //метод для доделывания оставшихся работ

Console.WriteLine("Остаток:");

IncrementCountDelegate operation;

for (int i = 0; i < restOfJobs.Count; i++)

{

operation = restOfJobs.Dequeue() as IncrementCountDelegate;

if (operation != null)

{

//вызов метода

operation.BeginInvoke((string)restOfJobs.Dequeue(), (int)restOfJobs.Dequeue(),null,null); ;

}

else { Console.WriteLine("Извлеченный элмент из стека не является операцией."); return; }

}

}