Многопоточное программирование

Часть 2: низкоуровневая многопоточность

Юрий Литвинов
yurii.litvinov@gmail.com

9

Примитивы синхронизации

- Лучше необходимости синхронизации вообще избегать
- Бывают:
  - User-mode — атомарные операции, реализующиеся на процессоре и не требующие участия планировщика
  - Kernel-mode — примитивы, управляющие тем, как поток обрабатывается планировщиком
    - Более тяжеловесные и медленные (до 1000 раз по сравнению с “без синхронизации вообще”)
    - Позволяют синхронизировать даже разные процессы

Атомарные операции

- Чтения и записи следующих типов всегда атомарны: Boolean, Char, (S)Byte, (U)Int16, (U)Int32, (U)IntPtr, Single, ссылочные типы
- Volatile
  - Volatile.Write
  - Volatile.Read
  - Связано с понятием Memory Fence, требует синхронизации ядер
  - Есть ключевое слово volatile: **private** volatile **int** flag = 0;
  - Volatile.Write должен быть последней операцией записи, Volatile.Read — первой операцией чтения

Пример

```
private int flag = 0;
private int value = 0;

public void Thread1() {
    value = 5;
    Volatile.Write(ref flag, 1);
}

public void Thread2() {
    if (Volatile.Read(ref flag) == 1)
        Console.WriteLine(value);
}
```

Interlocked

- Одновременные чтение и запись в одной “транзакции”
  - **public static** Int32 Increment(**ref** Int32 location);
  - **public static** Int32 Decrement(**ref** Int32 location);
  - **public static** Int32 Add(**ref** Int32 location, Int32 **value**);
  - **public static** Int32 Exchange(**ref** Int32 location, Int32 **value**);
  - **public static** Int32 CompareExchange(**ref** Int32 location, Int32 **value**, Int32 comparand);

Interlocked lock-free-операции

Compare-And-Swap loop

```
public static Int32 Maximum(ref Int32 target, Int32 value) {
   Int32 currentVal = target, startVal = 0, desiredVal = 0;
   do {
      startVal = currentVal;
      desiredVal = Math.Max(startVal, value);
      // Тут другой поток мог уже испортить target, так что если она изменилась,
      // надо начать всё сначала.
      currentVal = Interlocked.CompareExchange(ref target, desiredVal, startVal);
   } while (startVal != currentVal);
   return desiredVal;
}
```

Крутящееся ожидание

```
internal struct SimpleSpinLock {
    private int resourceInUse;

    public void Enter() {
        while (true) {
            if (Interlocked.Exchange(ref resourceInUse, 1) == 0)
                return;
        }
    }

    public void Leave() {
        Volatile.Write(ref resourceInUse, 0);
    }
}
```

Считается антипаттерном, но в некоторых ситуациях лучше lock-ов

Управление планировщиком

- Thread.Sleep(0) — ничего не делает, если остальные готовые потоки меньше приоритетом
- Thread.Sleep(1) — отдаёт управление потоку, даже если его приоритет меньше
- Thread.Yield() — нечто среднее (не вызовет переключения потоков, если желающих нет, в отличие от Thread.Sleep(1), но отдаст ядро потоку с меньшим приоритетом)
- Thread.SpinWait — просьба процессору с гипертредингом отдать ядро
- Очередной способ прострелить себе ногу — инверсия приоритетов
  - Поток с низким приоритетом захватил ресурс, нужный потоку с высоким приоритетом
  - Поток с высоким приоритетом крутится в ожидании, никогда не отдавая управление потоку, который мог бы отдать ресурс (livelock)

Примитивы уровня ядра ОС

- WaitHandle — всё, что можно ожидать
  - EventWaitHandle
    - AutoResetEvent — по сути, булевый флаг, поддерживаемый ОС
    - ManualResetEvent — тоже булевый флаг, но сбрасывается вручную
  - Semaphore — целое число, поддерживаемое ОС
  - Mutex — семафор, пропускающий только один поток

Пример (замок на Event-ах)

```
internal sealed class SimpleWaitLock : IDisposable {
   private readonly AutoResetEvent available;
   public SimpleWaitLock() {
      available = new AutoResetEvent(true);
   }

   public void Enter() {
      available.WaitOne();
   }

   public void Leave() {
      available.Set();
   }

   public void Dispose() { available.Dispose(); }
}
```

Пример (замок на семафорах)

```
public sealed class SimpleWaitLock : IDisposable {
   private readonly Semaphore available;
   public SimpleWaitLock(Int32 maxConcurrent) {
      available = new Semaphore(maxConcurrent, maxConcurrent);
   }

   public void Enter() {
      available.WaitOne();
   }

   public void Leave() {
      available.Release(1);
   }

   public void Dispose() { available.Close(); }
}
```

Пример (мьютекс, он сам замок)

```
internal class SomeClass : IDisposable {
   private readonly Mutex aLock = new Mutex();

   public void Method1() {
      aLock.WaitOne();
      Method2();  // Тут рекурсивный захват мьютекса
      aLock.ReleaseMutex();
   }

   public void Method2() {
      aLock.WaitOne();
      ...
      aLock.ReleaseMutex();
   }

   public void Dispose() { aLock.Dispose(); }
}
```

Гибридные конструкции

- ManualResetEventSlim
- SemaphoreSlim
- ReaderWriterLockSlim
- Monitor
  - Ключевое слово **lock**

Пример

```
internal sealed class Transaction {
   private readonly Object aLock = new Object();
   private DateTime timeOfLastTrans;

   public void PerformTransaction() {
      Monitor.Enter(aLock);
      timeOfLastTrans = DateTime.Now;
      Monitor.Exit(aLock);
   }

   public DateTime LastTransaction {
      get {
         Monitor.Enter(aLock);
         DateTime temp = timeOfLastTrans;
         Monitor.Exit(aLock);
         return temp;
      }
   }
}
```

lock

```
private void SomeMethod() {
    lock (this) {
        ...
    }
}
```

Так писать нельзя: любой, кто имеет ссылку на this, может взять на него замок без его ведома, что может привести к дедлоку, если замок взят из другого потока. Правильнее:

```
private Object lockObject = new Object();

private void SomeMethod() {
    lock (lockObject) {
        ...
    }
}
```

Литература

Must read

Jeffrey Richter, CLR via C# (4th Edition), Microsoft Press, 2012. 894pp.