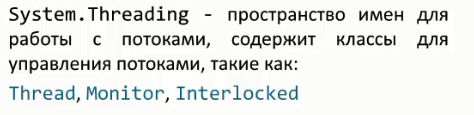


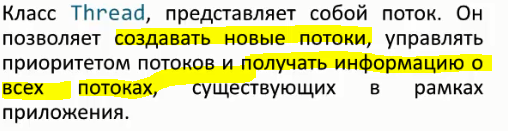
На основе процессов



На основе потоков







Thread

var thread = new Thread(new ThreadStart(ThreadFunc)); - Создание нового потока.

thread.Start(); - запуск потока

thread.Join(); - Ожидание первичным потоком, завершения работы вторичного потока.

Атрибут [ThreadStatic]

[ThreadStatic]

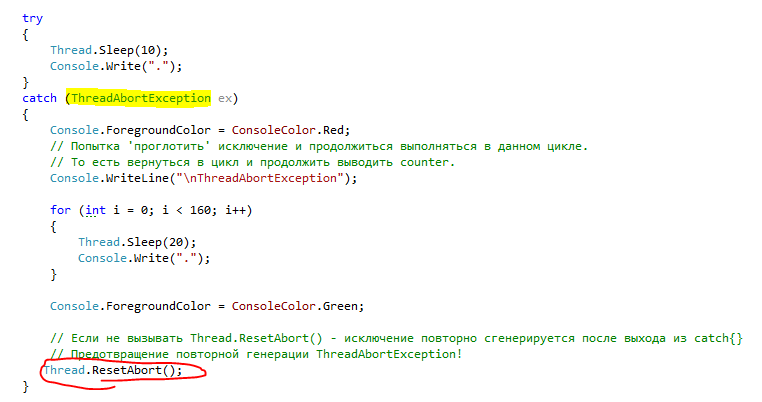
public static int counter; - Общая переменная счетчик для каждого созданного потока (в каждом потоке)

Отмена выполнения раннее запущенного потока

Abort();

ThreadAbortException

thread.Abort(); - Прервать поток. Генерируется исключение ThreadAbortException. В потоке thread. С целью отловить данное исключение необходимо в вторичном потоке реализовать блок catch (ThreadAbortException ex)



Thread.ResetAbort(); - Если не вызывать Thread.ResetAbort() - исключение повторно сгенерируется после выхода из catch{}(т.е. отмена метода .Abort())

а если не вызывать то после выхода из блока catch{} поток продолжит свое выполнение и не завершиться

Состояние потока

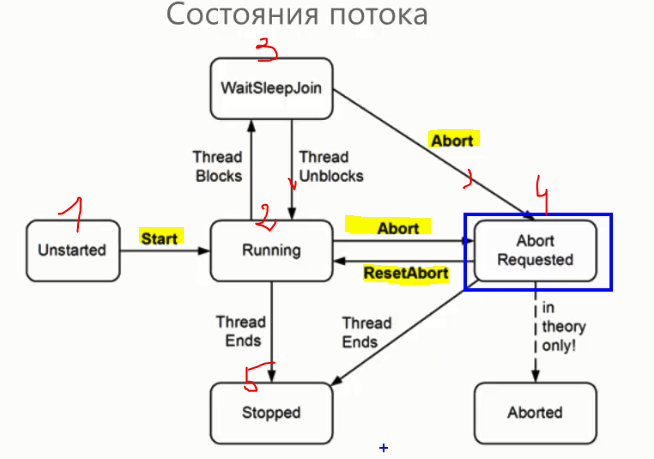
1 Unstarted до вызова start()

2 Running после вызова start()

3 WaitSleepJoin когда в потоке вызван метод Join() или Sleep() или уперся поток в критическую секцию lock{…..}

4 с состояния WaitSleepJoin может перейти в состояние Abort или Running

5 Abort из этого состояния может перейти в состояние Stopped



Есть два варианта работы потоков Foreground и Background

Foreground - Будет работать после завершения работы первичного потока.

Background - Завершает работу вместе с первичным потоком.

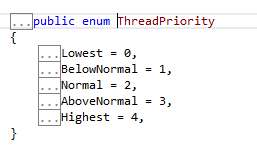
IsBackground - устанавливает поток как фоновый.

Не ждем завершения вторичного потока в данном случае.

По умолчанию - thread.IsBackground = false;

ThreadPriority

Приоритет потока



Interlocked

Предоставляет атомарные операции для переменных, общедоступных нескольким потокам. (на подобие критическое секции но ТОЛЬКО ДЛЯ ПЕРЕМЕННЫХ)

Interlocked.Increment(ref counter); - Поток увеличивает счетчик. (counter++;)

Interlocked.Decrement(ref counter); - Поток уменьшает счетчик.(counter--;)

long number = Interlocked.Read(ref counter); - считывание поля counter

int result = Interlocked.CompareExchange(ref block, 1, 0); - Метод CompareExchange()

1. Сравнивает начальное значение первого аргумента с третьим аргументом.

2. Если первый аргумент равен третьему аргументу, то в первый аргумент записывается значение второго аргумента.

3. Иначе, если первый аргумент не равен третьему аргументу, то первый аргумент остается без изменения.

4. Возвращает начальное значение первого аргумента (в любом случае).

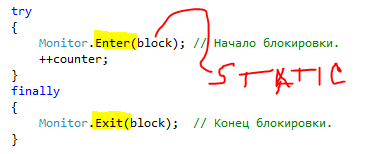
Interlocked.Exchange(ref block, 0); - записывает значение второго аргумента

Monitor

Monitor.Enter(block); - Начало блокировки.

Monitor.Exit(block); - Конец блокировки.

static private readonly object block = new object(); - Объект для блокировки.



lock

lock (block) – аналогично Monitor

{

++counter;

}

Преобразует в блок try finally в котором находится Monitor.Enter() и Monitor.Exit() как в примере сверху STATIC

Нельзя использовать объекты блокировки структурного типа.

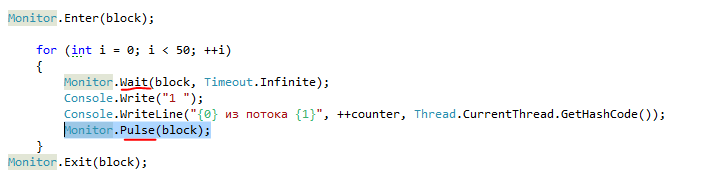
block - не может быть структурным.!!!!!!!!!!!!!!!!!!!!

static private int block = 0;

Monitor.Wait(block, Timeout.Infinite); - запускается в блоке lock (block) или после Monitor.Enter(block);

Monitor.Pulse(block); - посылает другому потоку сигнал (если поток использует такой же объект доступа) о том что он может зайти в критическую секцию для доступа к разделяемому ресурсу и остановиться на Monitor.Wait пока в том потоке не выйдет из lock (block), Monitor.Pulse(block); или Monitor.Exit(block);

Два одинаковых метода (ВСЕ ТРИ МЕТОДА ДОЛЖНЫ ОБЯЗАТЕЛЬНО ИСПОЛЬЗОВАТЬ ОДИНАКОВУЮ СЕКЦИЮ Monitor.Wait или lock (block))



А трети й метод первым идет Pulse()

