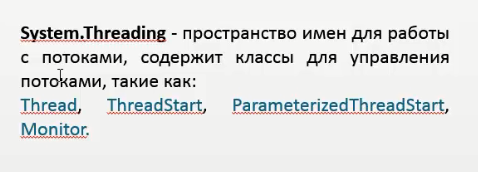


Многозадачность

На основе процессов – процесс может иметь один или более потоков, запущено много процессов (много программ)

На основе потоков – потоки выполняются в контексте одной программы (параллельное выполнение отдельных частей одной программы)

На каждый поток выделяется определенное количество процессорного кванта времени

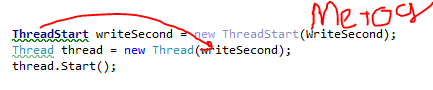


**ThreadStart – этот делегат принимает класс Thread (метод который будет сообщен с делегатом ничего не будет принимать и ничего не возвращать)**

**Thread - Запускает метод в контексте вторичного потока (асинхронно)**

**Синхронность – когда запускается метод в контексте первичного (текущего потока)**

**(**public delegate void **ThreadStart**();**)**



Для каждого потока выделяется 1 мегабайт стека



В связи с этим вторым параметром конструкторов передается размер стека, НО ЗАДАТЬ ЗНАЧЕНИЕ БОЛЬШЕЕ 1 МБ ИЛИ МЕНЬШЕЕ НЕУДАСТЬСЯ (РАЗМЕР СТЕКА ВСЕГДА РАВЕН 1 МБ)



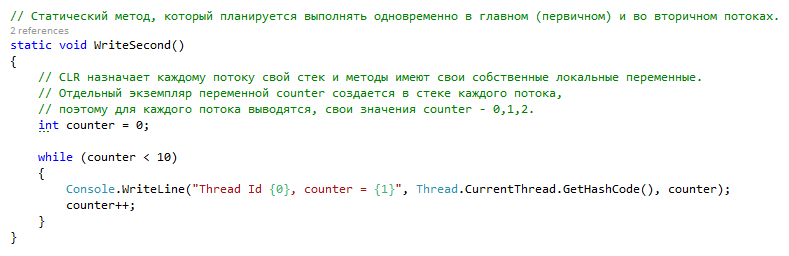
Переменные внутри методов не разделяемы и создаются внутри стека каждого потока

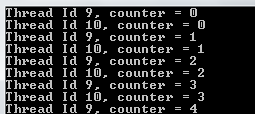
ПРИ РАБОТЕ С ПОТОКОМ СТРОИТСЯ КОПИЯ СТАТИЧЕСКОГО МЕТОДА (как и обычного метода)

CLR назначает каждому потоку свой стек и методы имеют свои собственные локальные переменные.

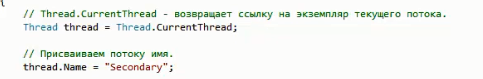
Отдельный экземпляр переменной counter создается в стеке каждого потока,

поэтому для каждого потока выводятся, свои значения counter - 0,1,2.

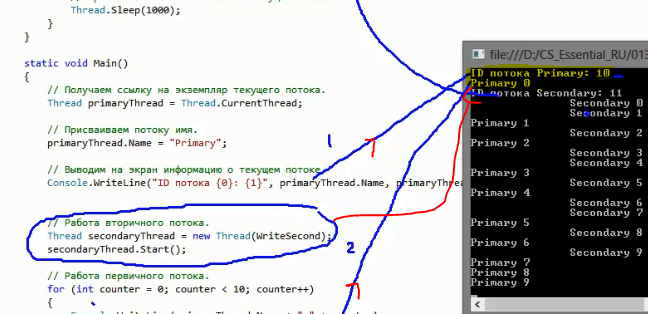




Можно получить текущий поток, и присвоить ему имя



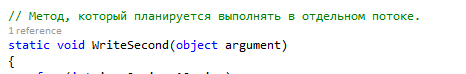
На построение потока уходит время (при старте он не запускается мгновенно) в связи с этим действие первичного потока продолжаются



**ParameterizedThreadStart – метод сообщенный с данным делегатом принимает один параметр, этот делегат принимает конструктор класса Thread и при запуске потока можно передать значение**



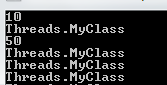
**Метод имеет один аргумент**

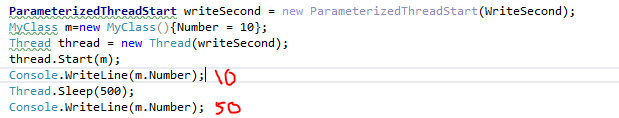


**ParameterizedThreadStart** writeSecond = new ParameterizedThreadStart(WriteSecond);

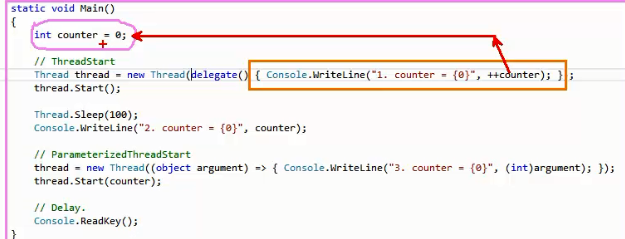


В случае передачи ссылочного типа (к примеру пользовательского класса) и изменении в вторичном потоке его свойства, то в первичном потоке оно также измениться





Замыкание – первая парадигма функционального программирования (когда мы можем обратиться к переменной находящегося в теле метода из тела другого метода )





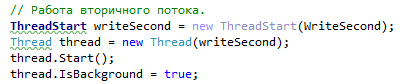
Основные и фоновые потоки.

По умолчанию свойство IsBackground равно false.

Чтобы вторичный поток не замораживал всю программу (если первичный поток завершиться)

То вторичному потоку нужно свойство IsBackground назначить true

Так как программа будет дожидаться окончания вторичного потока



Lock – критическая секция в которой может находиться один поток

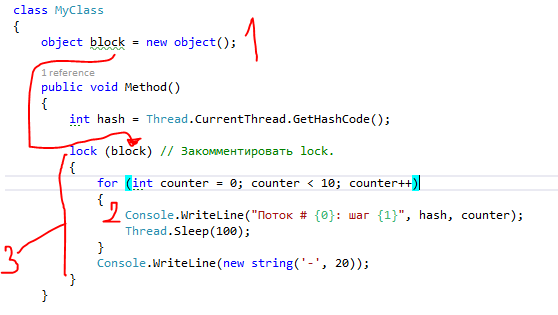
Техника синхронизации доступа к разделяемому ресурсу из нескольких потоков

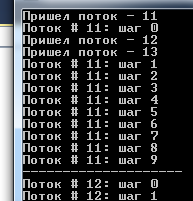
Потоки хотят получить доступ к разделяемому ресурсу

1 Объект синхронизации доступа к разделяемому ресурсу (объект блокировки)– следит за тем чтобы только один поток мог получить доступ к разделяемому ресурсу

2 Разделяемый ресурс – Console (к примеру)

3 Критическая секция – область кода где находится разделяемый ресурс





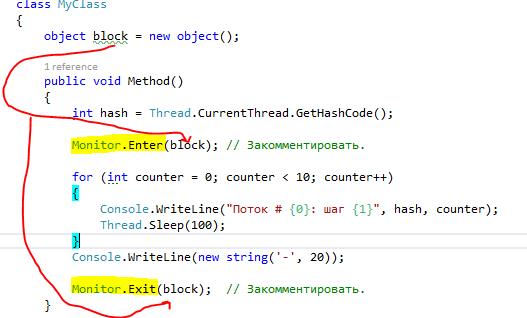
Monitor

Monitor.Enter – ЗАПИСЫВАЕТ В ОБЪЕКТ РАЗДЕЛЯЕМОГО РЕСУРСА NULL И ПОТОКИ ПРОВЕРЯЮТ, ЕСЛИ ОБЪЕКТ РАВЕН NULL ОНИ ЖДУТ

lock - это сокращенное использование System.Threading.Monitor.

Monitor.Enter(this) - блокирует блок кода так, что его может использовать только текущий поток.

Все остальные потоки ждут пока текущий поток, закончит работу и вызовет Monitor.Exit(this)



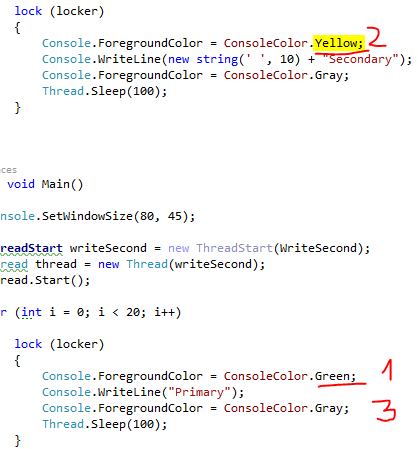
Monitor.Enter(this) – это заменяющая конструкция блока lock

В Monitor.Exit(this) ВСЕГДА НУЖНО ПЕРЕДАВАТЬ РАЗДЕЛЯЕМЫЙ ОБЪКТ КОТОРЫЙ ПЕРЕДАВАЛИ В Monitor.Enter(this), ИНАЧЕ ПОТОКИ ЖЕЛАЮЩИЕ ПОЛУЧИТЬ ДОСТУП К РАЗДЕЛЯЕМОМУ РЕСУРСУ НЕ СМОГУТ ЭТОГО СДЕЛАТЬ, Т.К. ОБЪЕКТ СИНХРОНИЗАЦИИ ДОСТУПА НЕ БУДЕТ ОСВОБОЖДЕН

Можно создать критические секции в разных местах (в случае если разделяемый ресурс один) тогда необходимо создать для них общий объект синхронизации доступа к ресурсу (объект блокировки)



На примере консоли когда (если закомментировать первый lock), то будет выполнение действий примерно такое, а может и другое, результат предугадать не получиться

  должно быть -