Потоки

№ урока: 11 **Курс:** C# Professional

Средства обучения: Компьютер с установленной Visual Studio

Обзор, цель и назначение урока

Урок посвящен основам работы с потоками средствами C#. Многопоточная программа состоит из двух или более частей, выполняемых параллельно. Каждая часть такой программы называется потоком и определяет отдельный путь выполнения команд. Таким образом, многопоточная обработка является особой формой многозадачности. С выпуском версии 4.0 в среде .NET Framework появились два важных дополнения, имеющих отношение к многопоточным приложениям, но понимание основ работы с потоками все же требуется для успешного сопровождения унаследованого кода и более глубокого понимания новшеств: PLINQ и TPL.

Изучив материал данного занятия, учащийся сможет:

- Понимать основы многопоточного программирования.
- Успешно применять возможности класса `Thread` в своей работе.
- Организовывать корректный запуск и завершение потоков.
- Осуществлять синхронизацию потоков.
- Приостанавливать либо завершать потоки в нужный момент времени.
- Использовать класс `Monitor` и конструкцию `lock`.

Содержание урока

1. Многозадачность. Многозадачность на основе процессов. Многозадачность на основе потоков.
2. Работа с потоками в .NET Framework: пространство имен `System.Threading`, класс `Thread`.
3. Создание потоков. Состояния потоков.
4. Завершение потоков. Определение момента окончания потока.
5. Передача аргументов потоку.
6. Использование `IsBackground`.
7. Приоритеты потоков.
8. Классы `Interlocked`, `Monitor`.
9. Синхронизация потоков.
10. Сообщение между потоками. Методы `Pulse()`, `Wait()`, `PulseAll()`.

Резюме

- Различают две разновидности многозадачности: на основе процессов и на основе потоков. В связи с этим важно понимать отличия между ними.
- Процесс фактически представляет собой исполняемую программу. Поэтому многозадачность на основе процессов — это средство, благодаря которому на компьютере могут параллельно выполняться две программы и более.
- Поток представляет собой координируемую единицу исполняемого кода. Своим происхождением этот термин обязан понятию "поток исполнения". При организации многозадачности на основе потоков у каждого процесса должен быть по крайней мере один поток, хотя их может быть и больше. Это означает, что в одной программе одновременно могут решаться две задачи и больше.
- Отличия в многозадачности на основе процессов и потоков могут быть сведены к следующему: многозадачность на основе процессов организуется для параллельного выполнения программ, а многозадачность на основе потоков — для параллельного выполнения отдельных частей одной программы.

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

- Поток может находиться в одном из нескольких состояний. В целом, поток может быть выполняющимся; готовым к выполнению, как только он получит время и ресурсы ЦП; приостановленным, т.е. временно не выполняющимся; возобновленным в дальнейшем; заблокированным в ожидании ресурсов для своего выполнения; а также завершенным, когда его выполнение окончено и не может быть возобновлено.
- В среде .NET Framework определены две разновидности потоков: приоритетный и фоновый. По умолчанию создаваемый поток автоматически становится приоритетным, но его можно сделать фоновым. Единственное отличие приоритетных потоков от фоновых заключается в том, что фоновый поток автоматически завершается, если в его процессе остановлены все приоритетные потоки.
- Классы, поддерживающие многопоточное программирование, определены в пространстве имен `System.Threading`. Система многопоточной обработки основывается на классе `Thread`, который инкапсулирует поток исполнения. Класс `Thread` является герметичным, т.е. он не может наследоваться.
- Для определения момента окончания потока можно воспользоваться значением свойства `IsAlive`. Оно вернет `false`, когда поток завершится. Кроме того, можно воспользоваться специальным методов `Join()`, который позволяет приостановить выполнение основного потока до момента завершения тех вторичных потоков, для которых метод `Join()` был использован.
- Первоначально в среде .NET Framework нельзя было передавать аргумент потоку, когда он начинался, поскольку у метода, служившего в качестве точки входа в поток, не могло быть параметров. Если же потоку требовалось передать какую-то информацию, то к этой цели приходилось идти различными обходными путями, например, использовать общую переменную. Но этот недостаток был впоследствии устранен, и теперь аргумент может быть передан потоку.
- Для вызова метода без аргументов в обычном потоке используется делегат **`ThreadStart`**, а для передачи аргументов в поток, необходимо использовать делегат **`ParameterizedThreadStart`**. С его помощью в поток можно передать один аргумент типа `object`.
- Для того чтобы сделать поток фоновым, достаточно присвоить логическое значение `true` свойству `IsBackground`. А логическое значение `false` указывает на то, что поток является приоритетным.
- У каждого потока имеется свой приоритет, который отчасти определяет, насколько часто поток получает доступ к ЦП. Вообще говоря, низкоприоритетные потоки получают доступ к ЦП реже, чем высокоприоритетные. Следует иметь в виду, что, помимо приоритета, на частоту доступа потока к ЦП оказывают влияние и другие факторы. Так, если высокоприоритетный поток ожидает доступа к некоторому ресурсу, например, для ввода с клавиатуры, он блокируется, а вместо него выполняется низкоприоритетный поток. В подобной ситуации низкоприоритетный поток может получать доступ к ЦП чаще, чем высокоприоритетный поток в течение определенного периода времени. И наконец, конкретное планирование задач на уровне операционной системы также оказывает влияние на время ЦП, выделяемое для потока.
- Приоритет потока можно изменить с помощью свойства `Priority`, являющегося членом класса `Thread`. По умолчанию для потока устанавливается значение приоритета `ThreadPriority.Normal`.
- Многопоточный код может вести себя по-разному в различных средах, поэтому никогда не следует полагаться на результаты его выполнения только в одной среде. Так, было бы ошибкой полагать, что низкоприоритетный поток из приведенного выше примера будет всегда выполняться лишь в течение небольшого периода времени до тех пор, пока не завершится высокоприоритетный поток. В другой среде высокоприоритетный поток может, например, завершиться еще до того, как низкоприоритетный поток выполнится хотя бы один раз.
- Когда используется несколько потоков, то иногда приходится координировать действия двух или более потоков. Процесс достижения такой координации называется синхронизацией. Самой распространенной причиной применения синхронизации служит необходимость разделять среди двух или более потоков общий ресурс, который может быть одновременно доступен только одному потоку.

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

- В основу синхронизации положено понятие блокировки, посредством которой организуется управление доступом к кодовому блоку в объекте. Когда объект заблокирован одним потоком, остальные потоки не могут получить доступ к заблокированному кодовому блоку. Когда же блокировка снимается одним потоком, объект становится доступным для использования в другом потоке.
- Итоги использования блокировки:
    - Если блокировка любого заданного объекта получена в одном потоке, то после блокировки объекта она не может быть получена в другом потоке.
    - Остальным потокам, пытающимся получить блокировку того же самого объекта, придется ждать до тех пор, пока объект не окажется в разблокированном состоянии.
    - Когда поток выходит из заблокированного фрагмента кода, соответствующий объект разблокируется.
- Синхронизация организуется с помощью ключевого слова `lock`.
- Ключевое слово `lock` на самом деле служит в C# быстрым способом доступа к средствам синхронизации, определенным в классе `Monitor`, который находится в пространстве имен `System.Threading`. В этом классе определен, в частности, ряд методов для управления синхронизацией. Например, для получения блокировки объекта вызывается метод `Enter`(), а для снятия блокировки — метод `Exit`().
- Методы `Wait`(), `Pulse`() и `PulseAll`() определены в классе `Monitor` и могут вызываться только из заблокированного фрагмента блока. Они применяются следующим образом. Когда выполнение потока временно заблокировано, он вызывает метод `Wait` (). В итоге поток переходит в состояние ожидания, а блокировка с соответствующего объекта снимается, что дает возможность использовать этот объект в другом потоке. В дальнейшем ожидающий поток активизируется, когда другой поток войдет в аналогичное состояние блокировки, и вызывает метод `Pulse`() или `PulseAll`(). При вызове метода `Pulse`() возобновляется выполнение первого потока, ожидающего своей очереди на получение блокировки. А вызов метода `PulseAll`() сигнализирует о снятии блокировки всем ожидающим потокам.
- Наиболее опасными проблемами, из тех, что возникают во время параллельного выполнения нескольких потоков являются проблемы взаимоблокировки и состояния гонки. Взаимоблокировка – это такое состояние потоков, когда каждый ждет окончания работы остальных и при этом ничего не делает. Состояние гонки – это попытки потоков получить доступ к одному ресурсу без должной его блокировки.

Закрепление материала

1. Что такое поток?
2. Какие виды многозадачности вы знаете? В чем между ними разница?
3. Зачем нужен класс `Thread`?
4. Как организовать передачу аргументов в поток?
5. Что такое фоновый поток?
6. Зачем нужна синхронизация потоков? Какими средствами ее можно осуществить?
7. Что такое взаимоблокировка?
8. В чем заключается основная проблема при возникновении состояния гонки? Как ее избежать?

Дополнительное задание

Используя конструкции блокировки, модифицируйте последний пример урока таким образом, чтобы получить возможность поочередной работы 3-х потоков.

Самостоятельная деятельность учащегося

Задание 1
Выучите основные конструкции и понятия, рассмотренные на уроке.

Задание 2

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

Создайте консольное приложение, которое в различных потоках сможет получить доступ к 2-м файлам. Считайте из этих файлов содержимое и попытайтесь записать полученную информацию в третий файл. Чтение/запись должны осуществляться одновременно в каждом из дочерних потоков. Используйте блокировку потоков для того, чтобы добиться корректной записи в конечный файл.

Задание 3
Зайдите на сайт MSDN.
Используя поисковые механизмы MSDN, найдите самостоятельно описание темы по каждому примеру, который был рассмотрен на уроке, так, как это представлено ниже, в разделе «Рекомендуемые ресурсы», описания данного урока. Сохраните ссылки и дайте им короткое описание.

Рекомендуемые ресурсы

MSDN: Пространство имен `System.Threading`
http://msdn.microsoft.com/ru-ru/library/798axes2.aspx

MSDN: Класс `Thread`
http://msdn.microsoft.com/ru-ru/library/system.threading.thread.aspx

MSDN: Класс `Monitor`
http://msdn.microsoft.com/ru-ru/library/system.threading.monitor.aspx

MSDN: Класс `Interlocked`
http://msdn.microsoft.com/ru-ru/library/system.threading.interlocked.aspx

MSDN: Оператор `lock`
http://msdn.microsoft.com/ru-ru/library/c5kehkcz.aspx

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com