Синхронизация при помощи объектов ядра и конструкций пользовательского режима.

№ урока: 12 **Курс:** C# Professional

Средства обучения: Компьютер с установленной Visual Studio

Обзор, цель и назначение урока

Синхронизация позволяет предотвратить повреждение общих данных при одновременном доступе к этим данным разных потоков. В данном уроке будут рассмотрены конструкции синхронизации пользовательского режима и режима ядра. Будет проведена сравнительная характеристика производительности приложения использующего синхронизацию.

Изучив материал данного занятия, учащийся сможет:

- Использовать конструкции синхронизации пользовательского режима.
- Использовать конструкции синхронизации режима ядра операционной системы.
- Применять гибридные конструкции синхронизации.
- Понимать преимущества и недостатки различных подходов синхронизации.

Содержание урока

1. Конструкции пользовательского режима
    a. Волатильные конструкции (volatile).
    b. Взаимозапирающие конструкции (interlocked).
2. Конструкции синхронизации режима ядра
    a. События `AutoResetEvent` и `ManualResetEvent`
    b. Семафоры `Semaphore`
    c. Мьютексы `Mutex`
3. Гибридные конструкции синхронизации потоков
    a. Monitor
    b. ManualResetEventSlim
    c. SemaphoreSlim

Резюме

- Синхронизация потоков является той процедурой, которую по возможности нужно избегать.
- Независимо от типа блокирующих объектов всегда нужно стремиться создавать код, в котором блокировка удерживается как можно меньше времени. Если, например, вам необходимо осуществить блокировку ресурса на долгое время, проверьте возможность создания локальной копии ресурса для дальнейшей работы с ним. Если такая возможность есть, удерживайте блокировку лишь для того, чтобы скопировать ресурс и продолжить работу с ним без установки длительной блокировки.
- Собственные библиотеки классов следует строить по следующему эталону: Все статические методы следует сделать безопасными в отношении потоков, а экземплярные – нет.
- Ключевое слово `volatile` указывает, что поле может быть изменено несколькими потоками, выполняющимися одновременно. Поля, объявленные как `volatile`, не проходят оптимизацию компилятором, которая предусматривает доступ посредством отдельного потока. Это гарантирует наличие наиболее актуального значения в поле в любое время.
- Ключевое слово `volatile` можно применять к полям следующих типов:
    - Ссылочные типы

CyberBionic systematics

CyberBionic Systematics ® 2015
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine

t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

    - Типы указателей (в небезопасном контексте). Обратите внимание, что несмотря на то, что сам указатель может быть `volatile`, объект, на который он указывает, таковым быть не может. Другими словами, нельзя объявить указатель `volatile`
    - Типы, такие как `sbyte`, `byte`, `short`, `ushort`, `int`, `uint`, `char`, `float` и `bool`.
    - Тип перечисления с одним из следующих базовых типов: `byte`, `sbyte`, `short`, `ushort`, `int` или `uint`.
    - Параметры универсальных типов, являющиеся ссылочными типами.
    - Свойства `IntPtr` и `UIntPtr`.
- Ключевое слово `volatile` можно применить только к полям класса или структуры. Локальные переменные не могут быть объявлены как `volatile`.
- `AutoResetEvent` позволяет потокам взаимодействовать друг с другом путем передачи сигналов. Как правило, этот класс используется, когда потокам требуется исключительный доступ к ресурсу.
- Вызов `Set` сигнализирует событию `AutoResetEvent` о необходимости освобождения ожидающего потока. Событие `AutoResetEvent` остается в сигнальном состоянии до освобождения одного ожидающего потока, а затем возвращается в несигнальное состояние. Если нет ожидающих потоков, состояние остается сигнальным бесконечно. Если поток вызывает метод `WaitOne`, а `AutoResetEvent` находится в сигнальном состоянии, поток не блокируется. `AutoResetEvent` немедленно освобождает поток и возвращается в несигнальное состояние.
- `ManualResetEvent` позволяет потокам взаимодействовать друг с другом путем передачи сигналов. Обычно это взаимодействие касается задачи, которую один поток должен завершить до того, как другой продолжит работу.
- Когда поток начинает работу, которая должна быть завершена до продолжения работы других потоков, он вызывает метод `Reset` для того, чтобы поместить `ManualResetEvent` в несигнальное состояние. Этот поток можно понимать как контролирующий `ManualResetEvent`. Потоки, которые вызывают метод `WaitOne` в `ManualResetEvent`, будут заблокированы, ожидая сигнала. Когда контролирующий поток завершит работу, он вызовет метод `Set` для сообщения о том, что ожидающие потоки могут продолжить работу. Все ожидающие потоки освобождаются.
- Используйте класс `Semaphore` для управления доступом к пулу ресурсов. Потоки производят вход в семафор, вызывая метод `WaitOne()`, унаследованный от класса `WaitHandle`, и освобождают семафор вызовом метода `Release()`.
- Счетчик на семафоре уменьшается на единицу каждый раз, когда в семафор входит поток, и увеличивается на единицу, когда поток освобождает семафор. Когда счетчик равен нулю, последующие запросы блокируются, пока другие потоки не освободят семафор. Когда семафор освобожден всеми потоками, счетчик имеет максимальное значение, заданное при создании семафора.
- Гарантированный порядок, в котором бы блокированные потоки входили в семафор, например, FIFO или LIFO, отсутствует.
- Семафоры, как и мьютексы, могут быть локальными и системными (именованными).
- Когда двум или более потокам одновременно требуется доступ к общему ресурсу, системе необходим механизм синхронизации, чтобы обеспечить использование ресурса только одним потоком одновременно. `Mutex` — примитив, который предоставляет эксклюзивный доступ к общему ресурсу только одному потоку синхронизации. Мьютекс также хранит информацию о потоке, который им владеет и количество блокировок который поток вызвал на мьютексе.
- Если поток завершается, владея мьютексом, то мьютекс называется брошенным. Состояние мьютекса задается сигнальным, и мьютекс переходит во владение следующему ожидающему потоку. Начиная с версии 2.0 платформы .NET Framework, в следующем потоке, получившем брошенный мьютекс, выдается исключение `AbandonedMutexException`. В версиях платформы .NET Framework 2.0 и более ранних исключение не выдавалась.
- Брошенный системный мьютекс может свидетельствовать о внезапном прекращении выполнения приложения (например, с помощью диспетчера задач Windows).
- Мьютексы бывают двух типов: локальные мьютексы (без имени), и именованные системные мьютексы. Локальный мьютекс существует только внутри одного процесса.

CyberBionic systematics

CyberBionic Systematics ® 2015
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine

t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

Он может использоваться любым потоком в процессе, который содержит ссылку на объект `Mutex`, представляющий мьютекс. Каждый неименованный объект `Mutex` представляет отдельный локальный мьютекс.

- Именованные системные мьютексы доступны в пределах всей операционной системы и могут быть использованы для синхронизации действий процессов. Можно создать объект `Mutex`, представляющий именованный системный мьютекс, используя конструктор с поддержкой имен. Объект операционной системы может быть создан в то же время, или существовать до создания объекта `Mutex`.

Закрепление материала

1. Что такое объект синхронизации уровня ядра Windows?
2. Зачем применяется ключевое слово `volatile`?
3. Каковы ограничения на использование `volatile`?
4. Что такое `Mutex`?
5. Что такое `Semaphore`?
6. Как использовать событийную блокировку?
7. Перечислите отличия между ручной и автоматической событийной блокировкой.

Дополнительное задание

Создайте `Semaphore`, осуществляющий контроль доступа к ресурсу из нескольких потоков. Организуйте упорядоченный вывод информации о получении доступа в специальный *.log файл.

Самостоятельная деятельность учащегося

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

Задание 2
Преобразуйте пример событийной блокировки таким образом, чтобы вместо ручной обработки использовалась автоматическая.

Задание 3
Создайте приложение, которое может быть запущено только в одном экземпляре (используя именованный `Mutex`).

Задание 4
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: Ключевое слово `volatile`
http://msdn.microsoft.com/ru-ru/library/x13ttww7.aspx

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

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

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

CyberBionic systematics
CyberBionic Systematics ® 2015
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

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

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

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

CyberBionic systematics

CyberBionic Systematics ® 2015
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine

t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com