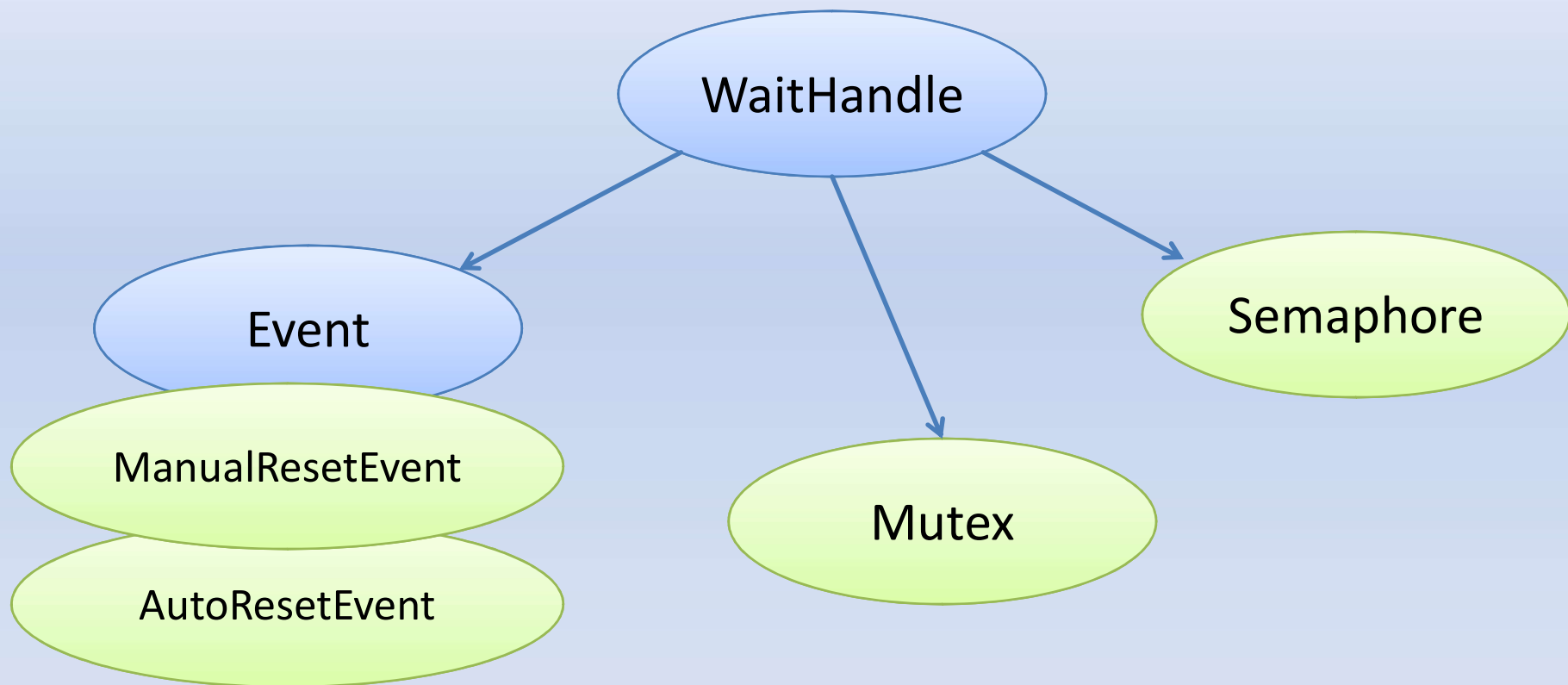


Объекты синхронизации ядра ОС



Mutex = mutual exclusion

- Объект Mutex используется для обеспечения взаимно-исключительного доступа к фрагменту кода (как и Monitor)
- В основе объекта Mutex лежат вызовы функций ядра операционной системы, поэтому блокировка является менее эффективной по сравнению с Monitor и lock
- Позволяет использовать глобальные именованные блокировки, доступные в рамках нескольких приложений
- Метод **WaitOne**: захват мьютекса = вход в критическую секцию
- Метод **Release**: освобождение мьютекса = выход из критической секции

```
class MyApplication
{
    static void Main()
    {
        // Именованный межпроцессный мьютекс
        var mutex = new Mutex(false, "MyApp ver 2.0");
        // Пытаемся захватить мьютекс в течении 5 сек
        if(!mutex.WaitOne(5000))
        {
            Console.WriteLine("Приложение уже запущено");
            return;
        }
        Run();
        mutex.Dispose();
    }
}
```

Сигнальная синхронизация

- Сигнальная (условная) синхронизация применяется для обеспечения определенного порядка выполнения потоков.
- Типовые сценарии сигнальной синхронизации: массовый старт, барьер, «эстафета»
- Средства сигнальной синхронизации на платформе .NET: `AutoResetEvent`, `ManualResetEvent`, `ManualResetEventSlim`, `Semaphore`, `SemaphoreSlim`, `CountdownEvent`, `Barrier`

```
void OneThread(object o)
{
    ManualResetEvent mre = (ManualResetEvent)o;
    mre.WaitOne();
    Console.WriteLine("Data from thread #2: " + data);
}
```


```
void SecondThread(object o)
{
    ManualResetEvent mre = (ManualResetEvent)o;
    Console.WriteLine("Writing data");
    data = "BBBBBB";
    mre.Set();
}
```

Сигнальные объекты

- Метод `WaitOne`: блокировка потока в ожидании сигнального состояния
- Метод `Set`: установка сигнального состояния и разблокировка потока (или потоков)
- Различие в сигнальных объектах заключается в режимах сброса:
 - Объект `AutoResetEvent` автоматически сбрасывает сигнальное состояние при разблокировке одного из ожидающих потоков
 - Объект `ManualResetEvent` сбрасывает сигнальное состояние только после вызова метода `Reset`
- Режим сброса определяет возможные применения сигнальных объектов: критическая секция, одновременный запуск нескольких потоков

Особенности сигнальных объектов

- AutoResetEvent гарантирует освобождение только одного потока при сигнальном состоянии
- Число вызовов Set \neq числу освобождаемых потоков
- Количество сигналов, сгенерированных до сброса объекта, не фиксируется
- Если несколько вызовов Set расположены достаточно близко, то может произойти освобождение меньшего числа потоков



```
event.Set();  
event.Set();
```

Thread #1

```
..  
event.WaitOne()  
..
```

Thread #2

```
..  
event.WaitOne()  
..
```

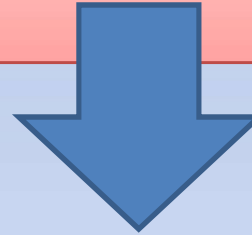




event.Set();
event.Set();

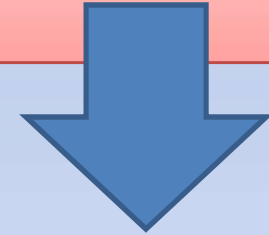
Thread #1

..
event.WaitOne()
..



Thread #2

..
event.WaitOne()
..

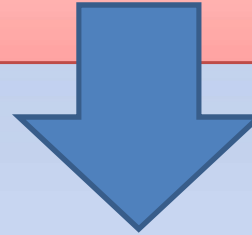




```
event.Set();  
Thread.Sleep(10);  
event.Set();
```

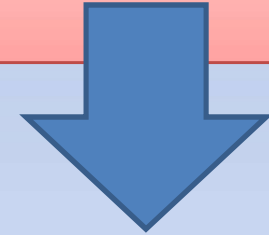
Thread #1

```
..  
event.WaitOne()  
..
```



Thread #2

```
..  
event.WaitOne()  
..
```

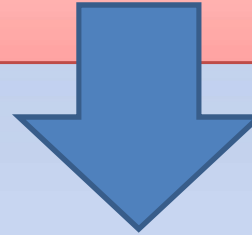




event.Set();

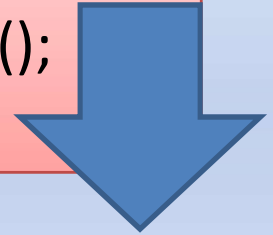
Thread #1

..
event.WaitOne()
..



Thread #2

..
event.WaitOne()
event.Set();
..



Семафоры

«A semaphore is like a nightclub: it has a certain capacity, enforced by a bouncer» J. Albahari

- Объект синхронизации с внутренним счетчиком, изменение которого выполняется потокобезопасно
- Метод WaitOne:
 - Если счетчик больше нуля – выполняется уменьшение счетчика, поток продолжает выполнение;
 - Если счетчик равен или меньше нуля, то поток блокируется;
- Метод Release:
 - Выполняется потокобезопасное увеличение счетчика и разблокировка одного из ожидающих потоков;
- При инициализации семафора можно определять начальное и максимальное значение внутреннего счетчика
- Семафор можно использовать для взаимно-исключительного доступа и для сигнальной синхронизации

```
// Инициализация семафора  
var sem = new Semaphore(3, 5);
```

```
..
```

```
sem.WaitOne();
```

```
..
```

```
sem.Release();
```

Семафоры в .net

- Работа с семафором потоконезависима (в отличие от mutex, lock): вызовы WaitOne и Release могут находиться в разных потоках
- Объект Semaphore использует вызовы функций ядра ОС и позволяет создавать именованные семафоры, существующие на уровне ОС (кросс-процессная синхронизация)
- `AutoResetEvent = Semaphore(0, 1)`
Двоичный семафор идентичен сигнальному объекту
- .NET предлагает «облегченный» вариант семафора – объект `SemaphoreSlim`, который реализует гибридную схему блокировки

Атомарные операторы

- Атомарные операторы предназначены для потокобезопасного неблокирующего выполнения примитивных операций над данными, преимущественно целочисленного типа
- «Атомарность» означает, что при выполнении операции одним потоком, другие потоки не вмешиваются
- Высокая эффективность из-за применения неблокирующих алгоритмов и специальных инструкций процессора (CAS-инструкции)
- Все атомарные операторы реализованы как статические методы класса `System.Threading.Interlocked`

Атомарные операторы

Оператор	Метод	Типы данных
Увеличение счетчика на единицу	Increment	Int32, Int64
Уменьшение счетчика на единицу	Decrement	Int32, Int64
Добавление	Add	Int32, Int64
Обмен значениями	Exchange	Int32, Int64, double, single, object
Условный обмен	CompareExchange	Int32, Int64, double, single, object
Чтение 64-разрядного целого	Read	Int64

// Пример 1

```
lock (sync_obj)
```

```
{ counter++; }
```

// можно выполнить с помощью атомарного оператора

```
Interlocked.Increment(ref counter);
```

// Пример 2

```
lock(LockObj)
```

```
{
```

```
    if(x == curVal)
```

```
        x = newVal;
```

```
}
```

```
oldVal = Interlocked.CompareExchange(ref x, newVal, curVal);
```