PV178: Programming for .NET Framework

Threads and Synchronization

Vojtěch Forejt, forejt@fi.muni.cz
Martin Osovský, osovsky@ics.muni.cz

Faculty of Informatics and **Institute of Computer Science**
Masaryk University

April 2, 2009

Threads

- Used to improve application's behaviour by running several tasks concurrently
  - Perform task in background while redrawing GUI
  - Download data from several servers at one time
  - ...
- `System.Threading` namespace
- `Thread` class provides basic thread support

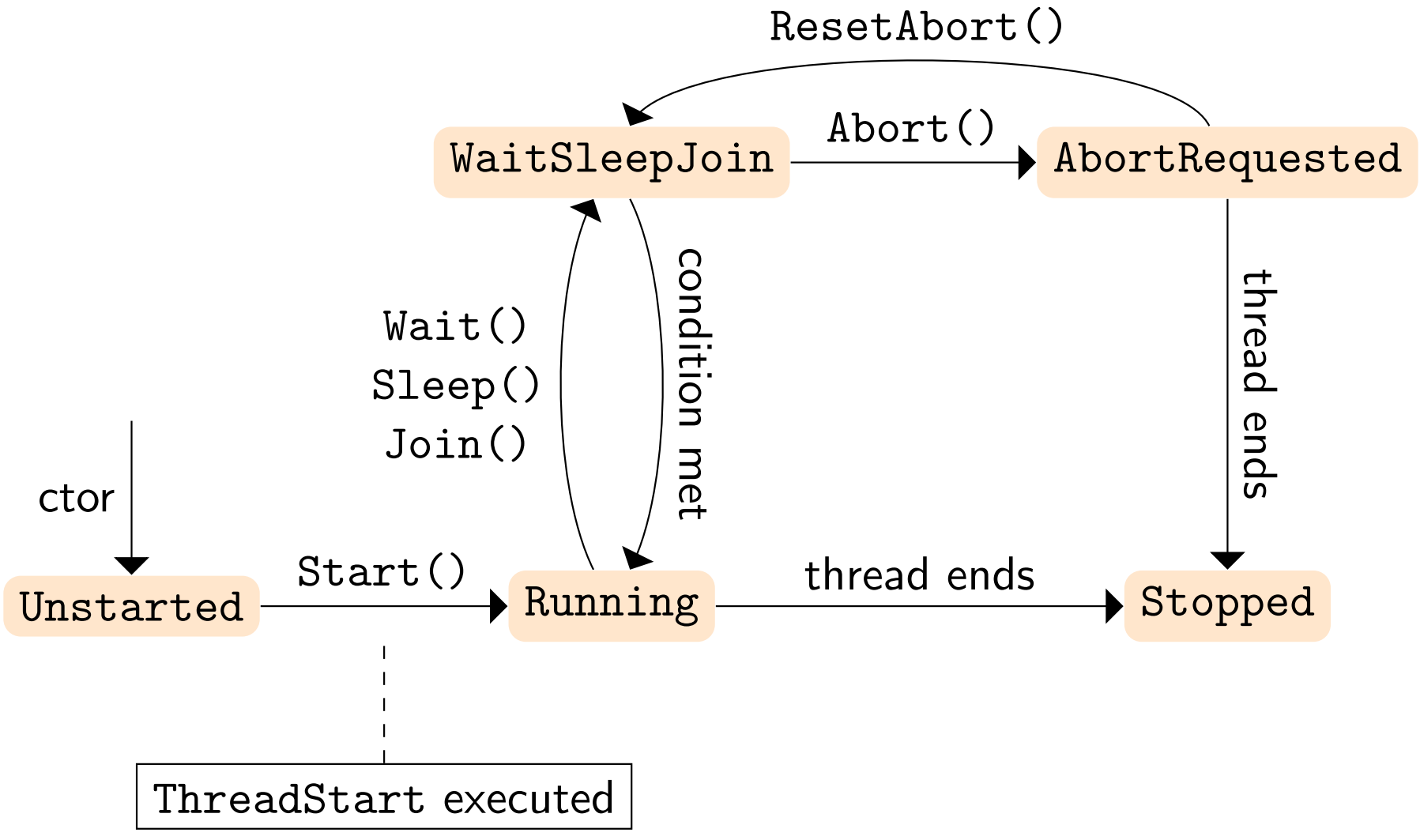
Thread Class

- Runs method given by delegate in constructor
  - delegate type: `ThreadStart`, `ParametrizedThreadStart`
- `Start` method starts the thread
- `ThreadState` property – `Unstarted`, `Running`, `WaitSleepJoin`, `AbortRequested`, `Stopped`,...
- Static method `Sleep` "blocks" the current thread for given time and changes thread state to `WaitSleepJoin`
- `Join` method "blocks" the current thread until a thread (whose `Join` method is called) terminates. Changes thread state to `WaitSleepJoin`

Thread Class cont.

- `Interrupt` method "wakes" the current thread
  - Raises `ThreadInterruptedException`
  - Changes thread state to `Running`
- `Abort` method may be used for terminating thread
  - `ThreadState` changed to `AbortRequested`
  - Raises `ThreadAbortException`
  - Called thread may call static method `ResetAbort` and state is changed to `Running`

Thread Life Cycle



Example

- ThreadsExample.cs

Thread Priority

- Specifies thread priority for scheduling purposes
- `Lowest`, `BelowNormal`, `Normal`, `AboveNormal`, `Highest`,

Example

- ThreadPriorityExample

`ThreadPool` Class

- Thread creation may be handled using this class;
- Delegate of type `WaitCallBack` is passed to static method `QueueUserWorkItem`.
- Maximal and minimal number of threads in thread pool may be get/set using `(Get|Set)(Min|Max)Threads`
- `GetAvailableThreads` returns number of threads that can be run in addition to current ones.

Example

- ThreadPoolExample.cs

Synchronization – Motivation

- In a multithreaded environment, some resources may be shared
- Access to shared resources may require special care

```
//two shared integers x and y
int z = y;
y = x;
x = z;
```

Example

- CriticalSectionExample.cs

Synchronization Primitives

- **Interlocked operations** – safely modify variables
- **Monitor** – gives access to a resource to one thread at a time
- **Mutex** – as above, allows synchronization of multiple processes
- **Semaphore** – gives acces to a resoucre to limited number of threads
- **ReaderWriterLock** – Allows multiple reading threads, or one writing thread

`Interlocked` Class

- Provides static methods to safely access shared variables
- Methods
  - `Increment` and `Decrement`
  - `Add`
  - `Exchange` and `CompareExchange` – allows exchanging values in variables
  - `Read` – reads a 64 bit value.

Monitor Class

- Controls access to objects
- Grants a lock for an object to a single thread
- Threads call monitor's static methods, object to synchronize on is given as parameter.
- Methods
  - `Enter` – acquire lock on object, if object is locked, blocks until it is freed
  - `TryEnter` – as above, but does not block and returns `bool`
  - `Exit` – releases the lock
  - `Wait` – releases the lock and blocks the thread until it reacquires the lock
  - `Pulse` – signals a waiting thread that object's lock state has changed
  - `PulseAll` – same as `Pulse`, but signals all waiting threads

Monitors – Syntactic Sugar

- In C#, `lock` statement performs `Enter` and `Exit` automatically

```
lock(obj)
{
  //now obj is locked for current thread
}
```

- `MethodImplAttribute` attribute of the method with value `Synchronized` marks critical section spanning whole method

```
[MethodImpl(MethodImplOptions.Synchronized)]
void SomeMethod()
{
  //this code will never be run by two threads
  //at the same time
}
```

Example

- MonitorsBasicExample

Example

- MonitorExample

`Mutex` Class

- May be used for synchronization between processes
- Methods
  - Static `OpenExisting` – returns mutex of a given name
  - `WaitOne`
  - `ReleaseMutex`

Example

- MutexExample

Semaphore Class

- Used where the number of threads accessing the shared resource should be limited
- Holds a count of threads currently in critical section
- Maximum number of threads in critical section given in constructor
- Methods
  - `WaitOne`
  - `Release`

Example

- SemaphoreExample.cs

`ReaderWriterLock` Class

- Used where a shared resource may be used for writing by one thread, or for reading by multiple threads.
- Methods
  - `AcquireReaderLock` and `AcquireReaderLock`
  - `UpgradeToWriterLock`
  - `ReleaseLock`

Asynchnous Operations

- Tasks that may be time consuming (e.g. IO) may support asynchronous calling.
- E.g. `BeginWrite` and `EndWrite` methods of `Stream`

`IAsyncResult` Interface

- Represent a status of asychronous operation, returned e.g. by `BeginWrite`
- Property `IsCompleted` – true if operation already ended

Calling Delegates Asynchronously

- Every delegate has the following methods
  - `BeginInvoke`
    - Takes same arguments as delegate, plus `AsyncCallback` delegate, and object representing application state (last two may be `null`)
    - Executed asynchronously, returns `IAsyncResult`
  - `EndInvoke`
    - Gets `IAsyncResult` and returns value returned by delegate calling
    - If asynchronous call has not ended yet, current thread blocks and waits for the end

Example

- AsyncDelegatesExample.cs