C# 멀티스레드 프로그래밍

최흥배
https://jacking75.github.io/

acorn+PACKT

C# 멀티스레드 프로그래밍 멀티코어를 위한 스레드, TPL, PLINQ, Rx 활용

유진 아가포노프 저 / 이문호 역 | 에이콘출판사 | 2016년 04월 14일 | 원서 : Multithreading in C# 5.0 Cookbook

5.0 회원리뷰(1건) | 판매지수 1032

정가	30,000원
판매가	**27,000원** (10% 할인)
할인혜택	24,000원 카카오뱅크 (3,000원 할인,캐시백, 3만원↑, 월1회)
	24,000원 BC체크카드 (3,000원 할인,쿠폰, 3만원↑)
	26,500원 PAYCO 페이코 (500원 할인,3천원↑, 최대3회)

판매중

수량 − 1 +

배송비 : 무료

카트에 넣기

바로구매

http://www.yes24.com/24/Goods/25466870?Acode=101

스레드 기초

스레드 생성

```
class Program
{
    static void Main(string[] args)
    {
        Thread t = new Thread(PrintNumbers);
        t.Start();
        PrintNumbers();
    }

    static void PrintNumbers()
    {
        Console.WriteLine("Starting...");
        for (int i = 1; i < 10; i++)
        {
            Console.WriteLine(i);
        }
    }
}
```

스레드 일시 정지

```
class Program
{
    static void Main(string[] args)
    {
        Thread t = new Thread(PrintNumbersWithDelay);
        t.Start();
        PrintNumbers();
    }

    static void PrintNumbers()
    {
        Console.WriteLine("Starting...");
        for (int i = 1; i < 10; i++)
        {
            Console.WriteLine(i);
        }
    }

    static void PrintNumbersWithDelay()
    {
        Console.WriteLine("Starting...");
        for (int i = 1; i < 10; i++)
        {
            Thread.Sleep(TimeSpan.FromSeconds(2));
            Console.WriteLine(i);
        }
    }
}
```

스레드 대기

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Starting program...");
        Thread t = new Thread(PrintNumbersWithDelay);
        t.Start();
        t.Join();
        Console.WriteLine("Thread completed");
    }

    static void PrintNumbersWithDelay()
    {
        Console.WriteLine("Starting...");
        for (int i = 1; i < 10; i++)
        {
            Thread.Sleep(TimeSpan.FromSeconds(2));
            Console.WriteLine(i);
        }
    }
}
```

스레드 중단

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Starting program...");
        Thread t = new Thread(PrintNumbersWithDelay);
        t.Start();
        Thread.Sleep(TimeSpan.FromSeconds(6));
        t.Abort();
        Console.WriteLine("A thread has been aborted");
    }

    static void PrintNumbersWithDelay()
    {
        Console.WriteLine("Starting...");
        for (int i = 1; i < 10; i++)
        {
            Thread.Sleep(TimeSpan.FromSeconds(2));
            Console.WriteLine(i);
        }
    }
}
```

스레드 상태 조사

```
static void PrintNumbersWithStatus()
{
    Console.WriteLine("Starting...");
    Console.WriteLine(Thread.CurrentThread.ThreadState.ToString());
    for (int i = 1; i < 10; i++)
    {
        Thread.Sleep(TimeSpan.FromSeconds(2));
        Console.WriteLine(i);
    }
```

스레드 우선순위

```
static void RunThreads()
{
    var sample = new ThreadSample();

    var threadOne = new Thread(sample.CountNumbers);
    threadOne.Name = "ThreadOne";
    var threadTwo = new Thread(sample.CountNumbers);
    threadTwo.Name = "ThreadTwo";

    threadOne.Priority = ThreadPriority.Highest;
    threadTwo.Priority = ThreadPriority.Lowest;
    threadOne.Start();
    threadTwo.Start();

    Thread.Sleep(TimeSpan.FromSeconds(2));
    sample.Stop();
}
```

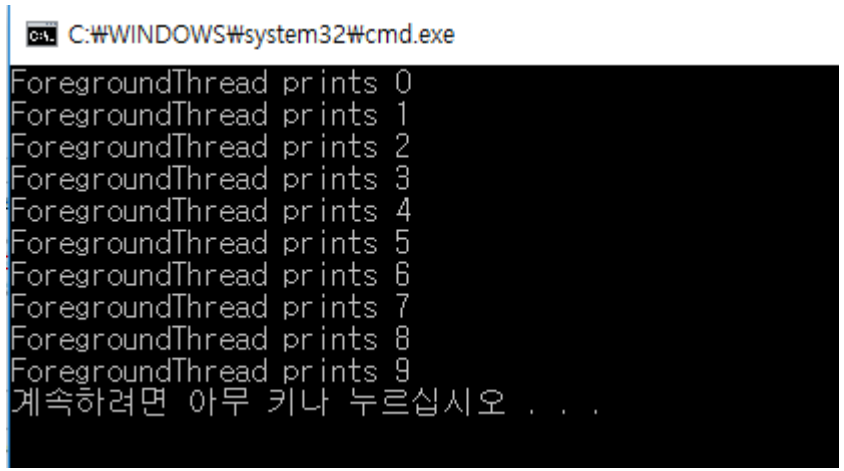
포그라운드 스레드와 백그라운드 스레드

프로세스는 작업을 완료하기 전에 모든 포그라운드 스레드가 끝나기를 기다리지만,
백그라운드 스레드가 있을 경우는 그냥 종료한다.

```
static void Main(string[] args)
{
    var sampleForeground = new ThreadSample(10);
    var sampleBackground = new ThreadSample(20);

    var threadOne = new Thread(sampleForeground.CountNumbers);
    threadOne.Name = "ForegroundThread";
    var threadTwo = new Thread(sampleBackground.CountNumbers);
    threadTwo.Name = "BackgroundThread";
    threadTwo.IsBackground = true;

    threadOne.Start();
    threadTwo.Start();
}

class ThreadSample
{
    private readonly int _iterations;

    public ThreadSample(int iterations)
    {
        _iterations = iterations;
    }
    public void CountNumbers()
    {
        for (int i = 0; i < _iterations; i++)
        {
            Thread.Sleep(TimeSpan.FromSeconds(0.5));
            Console.WriteLine("{0} prints {1}", Thread.CurrentThread.Name, i);
        }
    }
}
```

```
//threadOne.Start();
threadTwo.Start();
```


```
threadOne.Start();
//threadTwo.Start();
```



스레드에 파라미터 전달

```
static void Main(string[] args)
{
    var sample = new ThreadSample(10);

    var threadOne = new Thread(sample.CountNumbers);
    threadOne.Name = "ThreadOne";
    threadOne.Start();
    threadOne.Join();

    Console.WriteLine("--------------------------");

    var threadTwo = new Thread(Count);
    threadTwo.Name = "ThreadTwo";
    threadTwo.Start(8);
    threadTwo.Join();

    Console.WriteLine("--------------------------");

    var threadThree = new Thread(() => CountNumbers(12));
    threadThree.Name = "ThreadThree";
    threadThree.Start();
    threadThree.Join();
    Console.WriteLine("--------------------------");

    int i = 10;
    var threadFour = new Thread(() => PrintNumber(i));
    i = 20;
    var threadFive = new Thread(() => PrintNumber(i));
    threadFour.Start();
    threadFive.Start();
}
```

```
static void Count(object iterations)
{
    CountNumbers((int)iterations);
}

static void CountNumbers(int iterations)
{
    for (int i = 1; i <= iterations; i++)
    {
        Thread.Sleep(TimeSpan.FromSeconds(0.5));
        Console.WriteLine("{0} prints {1}", Thread.CurrentThread.Name, i);
    }
}

static void PrintNumber(int number)
{
    Console.WriteLine(number);
}

class ThreadSample
{
    private readonly int _iterations;

    public ThreadSample(int iterations)
    {
        _iterations = iterations;
    }
    public void CountNumbers()
    {
        for (int i = 1; i <= _iterations; i++)
        {
            Thread.Sleep(TimeSpan.FromSeconds(0.5));
            Console.WriteLine("{0} prints {1}", Thread.CurrentThread.Name, i);
        }
    }
}
```

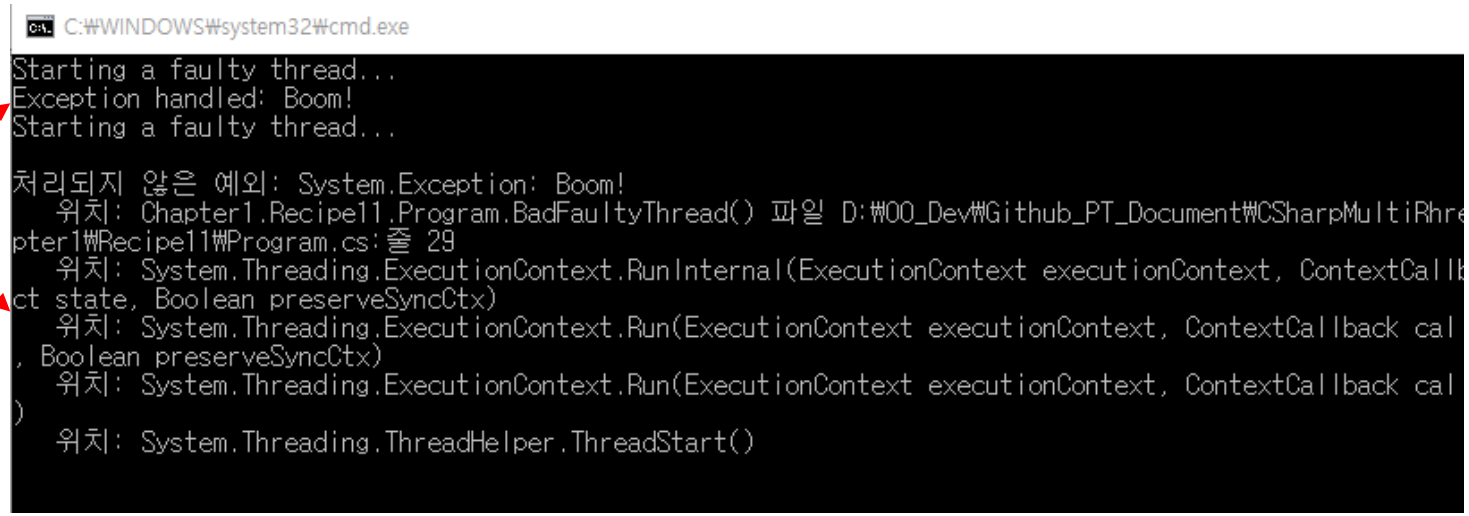
C#의 lock 키워드로 잠그기

```
class CounterWithLock : CounterBase
{
    private readonly object _syncRoot = new Object();

    public int Count { get; private set; }

    public override void Increment()
    {
        lock (_syncRoot)
        {
            Count++;
        }
    }

    public override void Decrement()
    {
        lock (_syncRoot)
        {
            Count--;
        }
    }
}
```

lock 키워드는 Monitor 클래스의 신텍스슈가 이다.

예외 처리

```
static void Main(string[] args)
{
    var t = new Thread(FaultyThread);
    t.Start();
    t.Join();

    try   NG
    {
        t = new Thread(BadFaultyThread);
        t.Start();
    }
    catch (Exception ex)
    {
        Console.WriteLine("We won't get here!");
    }
}

static void BadFaultyThread()
{
    Console.WriteLine("Starting a faulty thread...");
    Thread.Sleep(TimeSpan.FromSeconds(2));
    throw new Exception("Boom!");
}

static void FaultyThread()
{
    try   OK
    {
        Console.WriteLine("Starting a faulty thread...");
        Thread.Sleep(TimeSpan.FromSeconds(1));
        throw new Exception("Boom!");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Exception handled: {0}", ex.Message);
    }
}
```



스레드에서 실행하는 함수(FaultyThread) 안에서 예외를 잡아야 한다.

스레드 동기화

기본 원자 연산 수행

```
class CounterNoLock : CounterBase
{
    private int _count;

    public int Count { get { return _count; } }

    public override void Increment()
    {
        Interlocked.Increment(ref _count);
    }

    public override void Decrement()
    {
        Interlocked.Decrement(ref _count);
    }
}
```

Mutex 생성자 사용

```
static void Main(string[] args)
{
    const string MutexName = "CSharpThreadingCookbook";

    using (var m = new Mutex(false, MutexName))
    {
        if (!m.WaitOne(TimeSpan.FromSeconds(5), false))
        {
            Console.WriteLine("Second instance is running!");
        }
        else
        {
            Console.WriteLine("Running!");
            Console.ReadLine();
            m.ReleaseMutex();
        }
    }
}
```

SemaphoreSlim 생성자 사용

윈도우 커널 세마포어를 사용하지 않고, 내부 프로세스 동기화를 지원하지 않는다.
즉 프로세스간 동기화는 없으므로 프로세스간 동기화를 하고 싶다면 Semaphore를 사용한다.

```
static void Main(string[] args)
{
    for (int i = 1; i <= 6; i++)
    {
        string threadName = "Thread " + i;
        int secondsToWait = 2 + 2 * i;
        var t = new Thread(() => AccessDatabase(threadName, secondsToWait));
        t.Start();
    }
}

static SemaphoreSlim _semaphore = new SemaphoreSlim(4);

static void AccessDatabase(string name, int seconds)
{
    Console.WriteLine("{0} waits to access a database", name);
    _semaphore.Wait();
    Console.WriteLine("{0} was granted an access to a database", name);
    Thread.Sleep(TimeSpan.FromSeconds(seconds));
    Console.WriteLine("{0} is completed", name);
    _semaphore.Release();
}
```

AutoResetEvent 생성자 사용

대기 스레드에게 이벤트가 발생했음을 통지한다.

```
static void Main(string[] args)
{
    var t = new Thread(() => Process(10));
    t.Start();

    Console.WriteLine("Waiting for another thread to complete work");
    _workerEvent.WaitOne();
    Console.WriteLine("First operation is completed!");
    Console.WriteLine("Performing an operation on a main thread");
    Thread.Sleep(TimeSpan.FromSeconds(5));
    _mainEvent.Set();
    Console.WriteLine("Now running the second operation on a second thread");
    _workerEvent.WaitOne();
    Console.WriteLine("Second operation is completed!");
}

private static AutoResetEvent _workerEvent = new AutoResetEvent(false);
private static AutoResetEvent _mainEvent = new AutoResetEvent(false);

static void Process(int seconds)
{
    Console.WriteLine("Starting a long running work...");
    Thread.Sleep(TimeSpan.FromSeconds(seconds));
    Console.WriteLine("Work is done!");
    _workerEvent.Set();
    Console.WriteLine("Waiting for a main thread to complete its work");
    _mainEvent.WaitOne();
    Console.WriteLine("Starting second operation...");
    Thread.Sleep(TimeSpan.FromSeconds(seconds));
    Console.WriteLine("Work is done!");
    _workerEvent.Set();
}
```

ManualResetEventSlim 생성자 사용

Set()을 호출하면 스레드는 대기하지 않고, Reset()을 호출하면 스레드는 대기 상태에 들어간다.

```
static void Main(string[] args)
{
    var t1 = new Thread(() => TravelThroughGates("Thread 1", 5));
    var t2 = new Thread(() => TravelThroughGates("Thread 2", 6));
    var t3 = new Thread(() => TravelThroughGates("Thread 3", 12));
    t1.Start();
    t2.Start();
    t3.Start();
    Thread.Sleep(TimeSpan.FromSeconds(6));
    Console.WriteLine("The gates are now open!");
    _mainEvent.Set();
    Thread.Sleep(TimeSpan.FromSeconds(2));
    _mainEvent.Reset();
    Console.WriteLine("The gates have been closed!");
    Thread.Sleep(TimeSpan.FromSeconds(10));
    Console.WriteLine("The gates are now open for the second time!");
    _mainEvent.Set();
    Thread.Sleep(TimeSpan.FromSeconds(2));
    Console.WriteLine("The gates have been closed!");
    _mainEvent.Reset();
}

static void TravelThroughGates(string threadName, int seconds)
{
    Console.WriteLine("{0} falls to sleep", threadName);
    Thread.Sleep(TimeSpan.FromSeconds(seconds));
    Console.WriteLine("{0} waits for the gates to open!", threadName);
    _mainEvent.Wait();
    Console.WriteLine("{0} enters the gates!", threadName);
}

static ManualResetEventSlim _mainEvent = new ManualResetEventSlim(false);
```

CountdownEvent 생성자 사용

```
static void Main(string[] args)
{
    Console.WriteLine("Starting two operations");
    var t1 = new Thread(() => PerformOperation("Operation 1 is completed", 4));
    var t2 = new Thread(() => PerformOperation("Operation 2 is completed", 8));
    t1.Start();
    t2.Start();
    _countdown.Wait();
    Console.WriteLine("Both operations have been completed.");
    _countdown.Dispose();
}

static CountdownEvent _countdown = new CountdownEvent(2);

static void PerformOperation(string message, int seconds)
{
    Thread.Sleep(TimeSpan.FromSeconds(seconds));
    Console.WriteLine(message);
    _countdown.Signal();
}
```

Barrier 생성자 사용

```
    static void Main(string[] args)
    {
        var t1 = new Thread(() => PlayMusic("the guitarist", "play an amazing solo", 5));
        var t2 = new Thread(() => PlayMusic("the singer", "sing his song", 2));

        t1.Start();
        t2.Start();
    }

    static Barrier _barrier = new Barrier(2,
b => Console.WriteLine("End of phase {0}", b.CurrentPhaseNumber + 1));

    static void PlayMusic(string name, string message,
    {
        for (int i = 1; i < 3; i++)
        {
            Console.WriteLine("----------------------
            Thread.Sleep(TimeSpan.FromSeconds(seconds)
            Console.WriteLine("{0} starts to {1}", nam
            Thread.Sleep(TimeSpan.FromSeconds(seconds)
            Console.WriteLine("{0} finishes to {1}", n
            _barrier.SignalAndWait();
        }
    }
```

```
C:₩WINDOWS₩system32₩cmd.exe
----------------------------------------
----------------------------------------
the singer starts to sing his song
the singer finishes to sing his song
the guitarist starts to play an amazing solo
the guitarist finishes to play an amazing solo
End of phase 1
----------------------------------------
----------------------------------------
the singer starts to sing his song
the singer finishes to sing his song
the guitarist starts to play an amazing solo
the guitarist finishes to play an amazing solo
End of phase 2
계속하려면 아무 키나 누르십시오 . . .
```

지정된 횟수만큼 호출하면 등록된 콜백 함수를 호출한다

ReaderWriterLockSlim 생성자 사용

```
static ReaderWriterLockSlim _rw = new ReaderWriterLockSlim();
static Dictionary<int, int> _items = new Dictionary<int, int>();

static void Read()
{
    Console.WriteLine("Reading contents of a dictionary");
    while (true)
    {
        try
        {
            _rw.EnterReadLock();
            foreach (var key in _items.Keys)
            {
                Thread.Sleep(TimeSpan.FromSeconds(0.1));
            }
        }
        finally
        {
            _rw.ExitReadLock();
        }
    }
}
```

```
static void Write(string threadName)
{
    while (true)
    {
        try
        {
            int newKey = new Random().Next(250);
            _rw.EnterUpgradeableReadLock();
            if (!_items.ContainsKey(newKey))
            {
                try
                {
                    _rw.EnterWriteLock();
                    _items[newKey] = 1;
                    Console.WriteLine("New key {0} is
                }
                finally
                {
                    _rw.ExitWriteLock();
                }
            }
            Thread.Sleep(TimeSpan.FromSeconds(0.1));
        }
        finally
        {
            _rw.ExitUpgradeableReadLock();
        }
    }
}
```

SpinWait 생성자 사용

커널 모드와 무관하게 스레드에서 대기할 수 있다. 일정 시간 동안 사용자 모드에서 대기하다가 CPU 타임을 절약하기 위해 커널 모드로 전환한다.

[Thread.SpinWait와 SpinWait 구조체의 차이](http://pjc0247.tistory.com/12)
[Thread.Sleep()은 과연 얼마나 스레드를 재울까](http://blog.naver.com/vactorman/220181042697)

```
static void Main(string[] args)
{
    var t1 = new Thread(UserModeWait);
    var t2 = new Thread(HybridSpinWait);

    Console.WriteLine("Running user mode waiting");
    t1.Start();
    Thread.Sleep(20);
    _isCompleted = true;
    Thread.Sleep(TimeSpan.FromSeconds(1));
    _isCompleted = false;
    Console.WriteLine("Running hybrid SpinWait construct waiting");
    t2.Start();
    Thread.Sleep(5);
    _isCompleted = true;
}

static volatile bool _isCompleted = false;

static void UserModeWait()
{
    while (!_isCompleted)
    {
        Console.Write(".");
    }
    Console.WriteLine();
    Console.WriteLine("Waiting is complete");
}

static void HybridSpinWait()
{
    var w = new SpinWait();
    while (!_isCompleted)
    {
        w.SpinOnce();
        Console.WriteLine(w.NextSpinWillYield);
    }
    Console.WriteLine("Waiting is complete");
}
```

SpinLock

https://msdn.microsoft.com/ko-kr/library/dd997366(v=vs.110).aspx

스레드 풀 사용

스레드 풀에서 대리자 호출

APM 패턴

```
static void Main(string[] args)
{
    int threadId = 0;

    RunOnThreadPool poolDelegate = Test;

    var t = new Thread(() => Test(out threadId));
    t.Start();
    t.Join();

    Console.WriteLine("Thread id: {0}", threadId);

    IAsyncResult r = poolDelegate.BeginInvoke(out threadId, Callback, "a delegate asynchronous call");
    r.AsyncWaitHandle.WaitOne();

    string result = poolDelegate.EndInvoke(out threadId, r);

    Console.WriteLine("Thread pool worker thread id: {0}", threadId);
    Console.WriteLine(result);

    Thread.Sleep(TimeSpan.FromSeconds(2));
}

private delegate string RunOnThreadPool(out int threadId);

private static void Callback(IAsyncResult ar)
{
    Console.WriteLine("Starting a callback...");
    Console.WriteLine("State passed to a callbak: {0}", ar.AsyncState);
    Console.WriteLine("Is thread pool thread: {0}", Thread.CurrentThread.IsThreadPoolThread);
    Console.WriteLine("Thread pool worker thread id: {0}", Thread.CurrentThread.ManagedThreadId);
}
```

스레드 풀에 비동기 연산 넣기

```
static void Main(string[] args)
{
    const int x = 1;
    const int y = 2;
    const string lambdaState = "lambda state 2";

    ThreadPool.QueueUserWorkItem(AsyncOperation);
    Thread.Sleep(TimeSpan.FromSeconds(1));

    ThreadPool.QueueUserWorkItem(AsyncOperation, "async state");
    Thread.Sleep(TimeSpan.FromSeconds(1));

    ThreadPool.QueueUserWorkItem( state => {
            Console.WriteLine("Operation state: {0}", state);
            Console.WriteLine("Worker thread id: {0}", Thread.CurrentThread.ManagedThreadId);
            Thread.Sleep(TimeSpan.FromSeconds(2));
        }, "lambda state");

    ThreadPool.QueueUserWorkItem( _ =>
    {
        Console.WriteLine("Operation state: {0}, {1}", x+y, lambdaState);
        Console.WriteLine("Worker thread id: {0}", Thread.CurrentThread.ManagedThreadId);
        Thread.Sleep(TimeSpan.FromSeconds(2));
    }, "lambda state");

    Thread.Sleep(TimeSpan.FromSeconds(2));
}

private static void AsyncOperation(object state)
{
    Console.WriteLine("Operation state: {0}", state ?? "(null)");
    Console.WriteLine("Worker thread id: {0}", Thread.CurrentThread.ManagedThreadId);
    Thread.Sleep(TimeSpan.FromSeconds(2));
}
```

스레드 풀과 병렬도

스레드 작업이 시간이 많이 걸리면(IO 대기) 스레드 풀은 스레드 생성에 비해 느리다.

```
static void Main(string[] args)
{
    const int numberOfOperations = 500;
    var sw = new Stopwatch();
    sw.Start();
    UseThreads(numberOfOperations);
    sw.Stop();
    Console.WriteLine("Execution time using threads: {0}", sw.ElapsedMilliseconds);

    sw.Reset();
    sw.Start();
    UseThreadPool(numberOfOperations);
    sw.Stop();
    Console.WriteLine("Execution time using threads: {0}", sw.ElapsedMilliseconds);
}

static void UseThreads(int numberOfOperations)
{
    using (var countdown = new CountdownEvent(numberOfOperations))
    {
        Console.WriteLine("Scheduling work by creating threads");
        for (int i = 0; i < numberOfOperations; i++)
        {
            var thread = new Thread(() => {
                Console.Write("{0},", Thread.CurrentThread.ManagedThreadId);
                Thread.Sleep(TimeSpan.FromSeconds(0.1));
                countdown.Signal();
            });
            thread.Start();
        }
        countdown.Wait();
        Console.WriteLine();
    }
}

static void UseThreadPool(int numberOfOperations)
{
    using (var countdown = new CountdownEvent(numberOfOperations))
    {
        Console.WriteLine("Starting work on a threadpool");
        for (int i = 0; i < numberOfOperations; i++)
        {
            ThreadPool.QueueUserWorkItem( _ => {
                Console.Write("{0},", Thread.CurrentThread.ManagedThreadId);
                Thread.Sleep(TimeSpan.FromSeconds(0.1));
                countdown.Signal();
            });
        }
        countdown.Wait();
        Console.WriteLine();
    }
}
```

```
C:₩WINDOWS₩system32₩cmd.exe
Scheduling work by creating threads
3,6,4,7,5,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,2
5,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,6
5,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,1
119,120,121,122,123,124,125,126,127,128,129,130,131,132,1
149,150,151,152,153,154,155,156,157,158,159,160,161,162,1
179,180,181,182,183,184,185,186,187,188,189,190,191,192,1
209,210,211,212,213,214,215,216,217,218,219,220,221,222,2
239,240,241,242,243,244,245,246,247,248,249,250,251,252,2
269,270,271,272,273,274,275,276,277,278,279,280,281,282,2
299,300,301,302,303,304,305,306,307,308,309,310,311,312,3
329,330,331,332,333,334,335,336,337,338,339,340,341,342,3
359,360,361,362,363,364,365,366,367,368,369,370,371,372,3
389,390,391,392,393,394,395,396,397,398,399,400,401,402,4
419,420,421,422,423,424,425,426,427,428,429,430,431,432,4
449,450,451,452,453,454,455,456,457,458,459,460,461,462,4
479,480,481,482,483,484,485,486,487,488,489,490,491,492,4
Execution time using threads: 159
Starting work on a threadpool
505,503,507,504,510,506,508,509,505,507,504,503,510,506,5
509,508,504,503,505,506,510,507,509,508,504,503,505,507,5
508,509,504,505,507,511,503,510,508,509,506,504,505,511,5
505,507,503,510,511,506,509,504,508,505,507,503,510,511,5
507,511,505,506,504,508,509,510,503,507,511,504,505,506,5
511,505,508,509,503,510,507,506,511,504,505,508,509,510,5
505,503,506,511,507,508,505,504,509,503,508,507,511,504,5
505,509,508,503,506,511,507,509,508,504,505,503,511,506,5
507,509,508,504,505,503,511,507,506,509,508,504,505,503,5
507,511,509,506,508,504,505,503,510,511,507,509,508,506,5
509,508,505,504,503,510,511,507,506,509,508,505,504,503,5
505,510,503,507,511,508,509,506,504,505,507,503,510,511,5
511,508,503,504,506,509,505,510,507,508,511,503,504,506,5
506,509,505,503,510,511,508,504,503,509,506,505,511,510,5
511,510,506,509,505,503,508,504,510,511,506,503,505,509,5
508,510,504,505,511,503,506,509,508,510,504,511,505,506,5
503,509,508,510,504,505,506,511,503,509,507,508,512,510,5
Execution time using threads: 5948
계속하려면 아무 키나 누르십시오 . . .
```

취소 옵션 구현

```
using (var cts = new CancellationTokenSource())
{
    CancellationToken token = cts.Token;
    ThreadPool.QueueUserWorkItem(_ => AsyncOperation1(token));
    Thread.Sleep(TimeSpan.FromSeconds(2));
    cts.Cancel();
}
```

```
static void AsyncOperation1(CancellationToken token)
{
    Console.WriteLine("Starting the first task");
    for (int i = 0; i < 5; i++)
    {
        if (token.IsCancellationRequested)
        {
```

속성 확인 방법

```
static void AsyncOperation2(CancellationToken token)
{
    try
    {
        Console.WriteLine("Starting the second task");

        for (int i = 0; i < 5; i++)
        {
            token.ThrowIfCancellationRequested();
            Thread.Sleep(TimeSpan.FromSeconds(1));
        }
        Console.WriteLine("The second task has completed succesf
    }
    catch (OperationCanceledException)
    {
        Console.WriteLine("The second task has been canceled.");
    }
}
```

예외 던지기

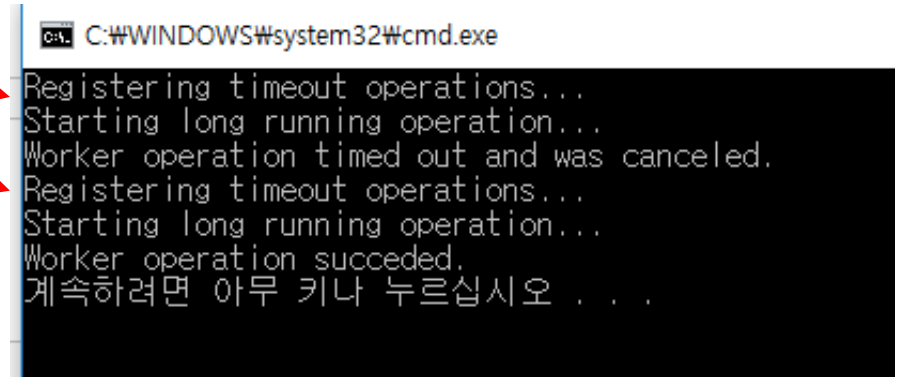
취소 되었을 때 스레드 풀에 호출되는 콜백 등록.

```
private static void AsyncOperation3(CancellationToken token)
{
    bool cancellationFlag = false;
    token.Register(() => cancellationFlag = true);
    Console.WriteLine("Starting the third task");
    for (int i = 0; i < 5; i++)
    {
        if (cancellationFlag)
        {
            Console.WriteLine("The third task has been canceled.");
            return;
```

스레드 풀을 이용한 대기 처리와 타임아웃 사용

작업이 끝나기 전에 타임아웃 시간 도달

```
static void Main(string[] args)
{
    RunOperations(TimeSpan.FromSeconds(5));
    RunOperations(TimeSpan.FromSeconds(7));
}

static void RunOperations(TimeSpan workerOperationTimeout)
{
    using (var evt = new ManualResetEvent(false))
    using (var cts = new CancellationTokenSource())
    {
        Console.WriteLine("Registering timeout operations...");
        var worker = ThreadPool.RegisterWaitForSingleObject(evt,
            (state, isTimedOut) => WorkerOperationWait(cts, isTimedOut), null, workerOperationTimeout, true);

        Console.WriteLine("Starting long running operation...");

        ThreadPool.QueueUserWorkItem(_ => WorkerOperation(cts.Token, evt));

        Thread.Sleep(workerOperationTimeout.Add(TimeSpan.FromSeconds(2)));
        worker.Unregister(evt);
    }
}

static void WorkerOperation(CancellationToken token, ManualResetEvent evt)
{
    for(int i = 0; i < 6; i++)
    {
        if (token.IsCancellationRequested)
        {
            return;
        }
        Thread.Sleep(TimeSpan.FromSeconds(1));
    }
    evt.Set();
}

static void WorkerOperationWait(CancellationTokenSource cts, bool isTimedOut)
{
    if (isTimedOut)
    {
        cts.Cancel();
        Console.WriteLine("Worker operation timed out and was canceled.");
    }
    else
    {
        Console.WriteLine("Worker operation succeded.");
    }
}
```



Recipe3-5

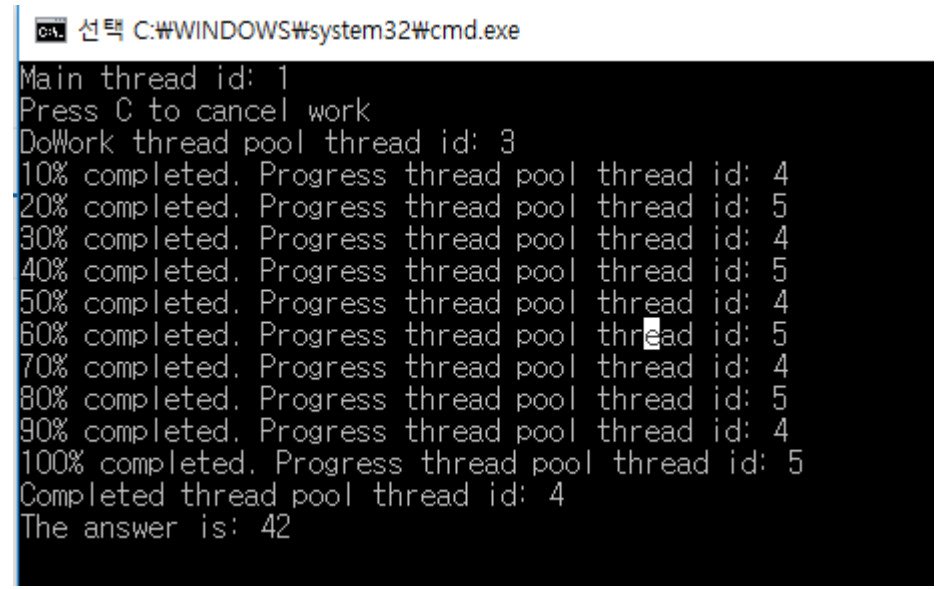
타이머 사용

```
static void Main(string[] args)
{
    Console.WriteLine("Press 'Enter' to stop the timer...");
    DateTime start = DateTime.Now;
    _timer = new Timer(_ => TimerOperation(start), null, TimeSpan.FromSeconds(1), TimeSpan.FromSeconds(2));

    Thread.Sleep(TimeSpan.FromSeconds(6));

    _timer.Change(TimeSpan.FromSeconds(1), TimeSpan.FromSeconds(4));

    Console.ReadLine();

    _timer.Dispose();
}

static Timer _timer;

static void TimerOperation(DateTime start)
{
    TimeSpan elapsed = DateTime.Now - start;
    Console.WriteLine("{0} seconds from {1}. Timer thread pool thread id: {2}", elapsed.Seconds, start,
        Thread.CurrentThread.ManagedThreadId);
}
```

BackgroundWorker 컴포넌트 사용

스레드 풀에서 동작한다.
Worker_DoWork, Worker_ProgressChanged, Worker_Completed 함수가 각각 다른 스레드에서 호출될 수 있다.

```
static void Main(string[] args)
{
    Console.WriteLine("Main thread id: {0}", Thread.CurrentThread.ManagedThreadId);

    var bw = new BackgroundWorker();
    bw.WorkerReportsProgress = true;
    bw.WorkerSupportsCancellation = true;

    bw.DoWork += Worker_DoWork;
    bw.ProgressChanged += Worker_ProgressChanged;
    bw.RunWorkerCompleted += Worker_Completed;

    bw.RunWorkerAsync();

    Console.WriteLine("Press C to cancel work");
    do
    {
        if (Console.ReadKey(true).KeyChar == 'C')
        {
            bw.CancelAsync();
        }

    }
    while(bw.IsBusy);
}

static void Worker_DoWork(object sender, DoWorkEventArgs e)
{
    Console.WriteLine("DoWork thread pool thread id: {0}", Thread.CurrentThread.ManagedThreadId);
    var bw = (BackgroundWorker) sender;
    for (int i = 1; i <= 100; i++)
    {

        if (bw.CancellationPending)
        {
            e.Cancel = true;
            return;
        }

        if (i%10 == 0)
        {
            bw.ReportProgress(i);
        }

        Thread.Sleep(TimeSpan.FromSeconds(0.1));
    }
    e.Result = 42;
}

static void Worker_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    Console.WriteLine("{0}% completed. Progress thread pool thread id: {1}", e.ProgressPercentage,
        Thread.CurrentThread.ManagedThreadId);
}

static void Worker_Completed(object sender, RunWorkerCompletedEventArgs e)
{
    Console.WriteLine("Completed thread pool thread id: {0}", Thread.CurrentThread.ManagedThreadId);
```



태스크 병렬 라이브러리 사용

비동기 프로그래밍의 불편함을 해결하기 위해 닷넷프레임워크 4.0 에서 태스크 병렬 라이브러리(TPL)이 생김.
TPL은 스레드 풀 위의 하나 이상의 추상 계층이며 스레드 풀을 이용한 작업을 하는 저수준 코드를 숨겨준다.

태스크 생성

```
static void Main(string[] args)
{
    var t1 = new Task(() => TaskMethod("Task 1"));
    var t2 = new Task(() => TaskMethod("Task 2"));
    t2.Start();
    t1.Start();
    Task.Run(() => TaskMethod("Task 3"));
    Task.Factory.StartNew(() => TaskMethod("Task 4"));
    Task.Factory.StartNew(() => TaskMethod("Task 5"), TaskCreationOptions.LongRunning);
    Thread.Sleep(TimeSpan.FromSeconds(1));
}

static void TaskMethod(string name)
{
    Console.WriteLine("Task {0} is running on a thread id {1}. Is thread pool thread: {2}",
        name, Thread.CurrentThread.ManagedThreadId, Thread.CurrentThread.IsThreadPoolThread);
}
```

생성된 태스크를 즉시 시작

옵션을 줄 수 있다.

Task.Run은 Task.Factory.StartNew의 축약 버전

태스크로 기본적인 연산 수행

```
static void Main(string[] args)
{
    TaskMethod("Main Thread Task");
    Task<int> task = CreateTask("Task 1");
    task.Start();
    int result = task.Result;
    Console.WriteLine("Result is: {0}", result);

    task = CreateTask("Task 2");
    task.RunSynchronously();
    result = task.Result;
    Console.WriteLine("Result is: {0}", result);

    task = CreateTask("Task 3");
    Console.WriteLine(task.Status);
    task.Start();

    while (!task.IsCompleted)
    {
        Console.WriteLine(task.Status);
        Thread.Sleep(TimeSpan.FromSeconds(0.5));
    }

    Console.WriteLine(task.Status);
    result = task.Result;
    Console.WriteLine("Result is: {0}", result);
}

static Task<int> CreateTask(string name)
{
    return new Task<int>(() => TaskMethod(name));
}

static int TaskMethod(string name)
{
    Console.WriteLine("Task {0} is running on a thread id {1}. Is thread pool thread: {2}",
        name, Thread.CurrentThread.ManagedThreadId, Thread.CurrentThread.IsThreadPoolThread);
    Thread.Sleep(TimeSpan.FromSeconds(2));
    return 42;
}
```

태스크를 함께 조합

```
static int TaskMethod(string name, int seconds)
{
    Console.WriteLine("Task {0} is running on a thread id {1}. Is thread pool thread: {2}",
        name, Thread.CurrentThread.ManagedThreadId, Thread.CurrentThread.IsThreadPoolThread);
    Thread.Sleep(TimeSpan.FromSeconds(seconds));
    return 42 * seconds;
}
```

```
var firstTask = new Task<int>(() => TaskMethod("First Task", 3));
var secondTask = new Task<int>(() => TaskMethod("Second Task", 2));

firstTask.ContinueWith(
    t => Console.WriteLine("The first answer is {0}. Thread id {1}, is thread pool thread: {2}",
        t.Result, Thread.CurrentThread.ManagedThreadId, Thread.CurrentThread.IsThreadPoolThread),
    TaskContinuationOptions.OnlyOnRanToCompletion);

firstTask.Start();
secondTask.Start();
```

```
Task continuation = secondTask.ContinueWith(
    t => Console.WriteLine("The second answer is {0}. Thread id {1}, is thread pool thread: {2}",
        t.Result, Thread.CurrentThread.ManagedThreadId, Thread.CurrentThread.IsThreadPoolThread),
    TaskContinuationOptions.OnlyOnRanToCompletion | TaskContinuationOptions.ExecuteSynchronously);

continuation.GetAwaiter().OnCompleted(
    () => Console.WriteLine("Continuation Task Completed! Thread id {0}, is thread pool thread: {1}",
        Thread.CurrentThread.ManagedThreadId, Thread.CurrentThread.IsThreadPoolThread));
```

```
firstTask = new Task<int>(() =>
{
    var innerTask = Task.Factory.StartNew(() => TaskMethod("Second Task", 5), TaskCreationOptions.AttachedToParent);
    innerTask.ContinueWith(t => TaskMethod("Third Task", 2), TaskContinuationOptions.AttachedToParent);
    return TaskMethod("First Task", 2);
});

firstTask.Start();
```

APM 패턴을 태스크로 변환

```
private delegate string AsynchronousTask(string threadName);
private delegate string IncompatibleAsynchronousTask(out int threadId);
```

```
int threadId;
AsynchronousTask d = Test;
IncompatibleAsynchronousTask e = Test;

Console.WriteLine("Option 1");
Task<string> task = Task<string>.Factory.FromAsync(
    d.BeginInvoke("AsyncTaskThread", Callback, "a delegate asynchronous call"), d.EndInvoke);

task.ContinueWith(t => Console.WriteLine("Callback is finished, now running a continuation! Result: {0}",(3)
    t.Result));
```

```
private static void Callback(IAsyncResult ar)  (2)
{
    Console.WriteLine("Starting a callback...");
    Console.WriteLine("State passed to a callbak: {0}", ar.AsyncState);
    Console.WriteLine("Is thread pool thread: {0}", Thread.CurrentThread.IsThreadPoolThread);
    Console.WriteLine("Thread pool worker thread id: {0}", Thread.CurrentThread.ManagedThreadId);
}

private static string Test(string threadName)
{
    Console.WriteLine("Starting...");  (1)
    Console.WriteLine("Is thread pool thread: {0}", Thread.CurrentThread.IsThreadPoolThread);
    Thread.Sleep(TimeSpan.FromSeconds(2));
    Thread.CurrentThread.Name = threadName;
    return string.Format("Thread name: {0}", Thread.CurrentThread.Name);
}
```

```
Option 1
Starting...
Is thread pool thread: True
WaitingForActivation
WaitingForActivation
WaitingForActivation
WaitingForActivation
Starting a callback...
State passed to a callbak: a delegate asynchronous call
Is thread pool thread: True
Thread pool worker thread id: 3
Callback is finished, now running a continuation! Result: Thread name: AsyncTaskThread
RanToCompletion
----------------------------------------
```

```
task = Task<string>.Factory.FromAsync(
    d.BeginInvoke, d.EndInvoke, "AsyncTaskThread", "a delegate asynchronous call");
task.ContinueWith(t => Console.WriteLine("Task is completed, now running a continuation! Result: {0}",
    t.Result));
```

콜백을 사용하지 않는 버전

```
Option 2
WaitingForActivation
Starting...
Is thread pool thread: True
WaitingForActivation
WaitingForActivation
WaitingForActivation
Task is completed, now running a continuation! Result: Thread name: AsyncTaskThread
RanToCompletion
```

```
IAsyncResult ar = e.BeginInvoke(out threadId, Callback, "a delegate asynchronous call");
task = Task<string>.Factory.FromAsync(ar, _ => e.EndInvoke(out threadId, ar));
task.ContinueWith(t =>
    Console.WriteLine("Task is completed, now running a continuation! Result: {0}, ThreadId: {1}",
        t.Result, threadId));
```

```
Option 3
WaitingForActivation
Starting...
Is thread pool thread: True
WaitingForActivation
WaitingForActivation
WaitingForActivation
Starting a callback...
State passed to a callbak: a delegate asynchronous call
Is thread pool thread: True
Thread pool worker thread id: 4
Task is completed, now running a continuation! Result: Thread pool worker thread id was: 4, ThreadId: 4
RanToCompletion
```

EAP 패턴을 태스크로 변환

```
static void Main(string[] args)
{
    var tcs = new TaskCompletionSource<int>();

    var worker = new BackgroundWorker();
    worker.DoWork += (sender, eventArgs) =>
    {
        eventArgs.Result = TaskMethod("Background worker", 5);
    };

    worker.RunWorkerCompleted += (sender, eventArgs) =>
    {
        if (eventArgs.Error != null)
        {
            tcs.SetException(eventArgs.Error);
        }
        else if (eventArgs.Cancelled)
        {
            tcs.SetCanceled();
        }
        else
        {
            tcs.SetResult((int)eventArgs.Result);
        }
    };

    worker.RunWorkerAsync();

    int result = tcs.Task.Result;

    Console.WriteLine("Result is: {0}", result);
}

static int TaskMethod(string name, int seconds)
{
    Console.WriteLine("Task {0} is running on a thread id {1}. Is thread pool thread: {2}",
        name, Thread.CurrentThread.ManagedThreadId, Thread.CurrentThread.IsThreadPoolThread);
    Thread.Sleep(TimeSpan.FromSeconds(seconds));
    return 42 * seconds;
}
```

취소 옵션 구현

```
private static void Main(string[] args)
{
    var cts = new CancellationTokenSource();
    var longTask = new Task<int>(() => TaskMethod("Task 1", 10, cts.Token), cts.Token);
    Console.WriteLine(longTask.Status);
    cts.Cancel();
    Console.WriteLine(longTask.Status);
    Console.WriteLine("First task has been cancelled before execution");
    cts = new CancellationTokenSource();
    longTask = new Task<int>(() => TaskMethod("Task 2", 10, cts.Token), cts.Token);
    longTask.Start();
    for (int i = 0; i < 5; i++)
    {
        Thread.Sleep(TimeSpan.FromSeconds(0.5));
        Console.WriteLine(longTask.Status);
    }
    cts.Cancel();
    for (int i = 0; i < 5; i++)
    {
        Thread.Sleep(TimeSpan.FromSeconds(0.5));
        Console.WriteLine(longTask.Status);
    }

    Console.WriteLine("A task has been completed with result {0}.", longTask.Result);
}

private static int TaskMethod(string name, int seconds, CancellationToken token)
{
    Console.WriteLine("Task {0} is running on a thread id {1}. Is thread pool thread: {2}",
        name, Thread.CurrentThread.ManagedThreadId, Thread.CurrentThread.IsThreadPoolThread);
    for (int i = 0; i < seconds; i ++)
    {
        Thread.Sleep(TimeSpan.FromSeconds(1));
        if (token.IsCancellationRequested) return -1;
    }
    return 42*seconds;
}
```

cst를 두번이나 사용하는 이유는 태스크를 시작전에 취소하는 경우에 TPL이 취소 처리를 하도록 하기 위함이다.

태스크 시작 전에 취소한 후 태스크를 시작하면 예외가 발생한다.

태스크를 병렬로 실행

```
static void Main(string[] args)
{
    var firstTask = new Task<int>(() => TaskMethod("First Task", 3));
    var secondTask = new Task<int>(() => TaskMethod("Second Task", 2));
    var whenAllTask = Task.WhenAll(firstTask, secondTask);

    whenAllTask.ContinueWith(t =>
        Console.WriteLine("The first answer is {0}, the second is {1}", t.Result[0], t.Result[1]),
        TaskContinuationOptions.OnlyOnRanToCompletion
        );

    firstTask.Start();
    secondTask.Start();

    Thread.Sleep(TimeSpan.FromSeconds(4));

    var tasks = new List<Task<int>>();
    for (int i = 1; i < 4; i++)
    {
        int counter = i;
        var task = new Task<int>(() => TaskMethod(string.Format("Task {0}", counter), counter));
        tasks.Add(task);
        task.Start();
    }

    while (tasks.Count > 0)
    {
        var completedTask = Task.WhenAny(tasks).Result;
        tasks.Remove(completedTask);
        Console.WriteLine("A task has been completed with result {0}.", completedTask.Result);
    }

    Thread.Sleep(TimeSpan.FromSeconds(1));
}

static int TaskMethod(string name, int seconds)
{
    Console.WriteLine("Task {0} is running on a thread id {1}. Is thread pool thread: {2}",
        name, Thread.CurrentThread.ManagedThreadId, Thread.CurrentThread.IsThreadPoolThread);
    Thread.Sleep(TimeSpan.FromSeconds(seconds));
    return 42 * seconds;
}
```

네이티브 비동기 프로그래밍

(async/await)

비동기 태스크 결과를 얻는 await 연산자 사용

```
async static Task AsynchronyWithAwait()
{
    try
    {
        string result = await GetInfoAsync("Task 2");
        Console.WriteLine(result);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex);
    }
}

async static Task<string> GetInfoAsync(string name)
{
    await Task.Delay(TimeSpan.FromSeconds(2));
    //throw new Exception("Boom!");
    return string.Format("Task {0} is running on a thread id {1}. Is thread pool thread: {2}",
        name, Thread.CurrentThread.ManagedThreadId, Thread.CurrentThread.IsThreadPoolThread);
}
```

람다 표현식에서 await 연산자 사용

```
static void Main(string[] args)
{
    Task t = AsynchronousProcessing();
    t.Wait();
}

async static Task AsynchronousProcessing()
{
    Func<string, Task<string>> asyncLambda = async name => {
        await Task.Delay(TimeSpan.FromSeconds(2));
        return string.Format("Task {0} is running on a thread id {1}. Is thread pool thread: {2}",
            name, Thread.CurrentThread.ManagedThreadId, Thread.CurrentThread.IsThreadPoolThread);
    };

    string result = await asyncLambda("async lambda");

    Console.WriteLine(result);
}
```

병렬 비동기 태스크 실행을 위한 await 연산자 사용

```
static void Main(string[] args)
{
    Task t = AsynchronousProcessing();
    t.Wait();
}

async static Task AsynchronousProcessing()
{
    Task<string> t1 = GetInfoAsync("Task 1", 3);
    Task<string> t2 = GetInfoAsync("Task 2", 5);

    string[] results = await Task.WhenAll(t1, t2);
    foreach (string result in results)
    {
        Console.WriteLine(result);
    }
}

async static Task<string> GetInfoAsync(string name, int seconds)
{
    await Task.Delay(TimeSpan.FromSeconds(seconds)); 같은 스레드에서 처리. 적은 스레드로 많은 일을 할 수 있음
    //await Task.Run(() => Thread.Sleep(TimeSpan.FromSeconds(seconds))); 다른 스레드에서 처리
    return string.Format("Task {0} is running on a thread id {1}. Is thread pool thread: {2}",
        name, Thread.CurrentThread.ManagedThreadId, Thread.CurrentThread.IsThreadPoolThread);
}
```

비동기 연산에서 예외 처리

```
Task<string> t1 = GetInfoAsync("Task 1", 3);
Task<string> t2 = GetInfoAsync("Task 2", 2);
try
{
    string[] results = await Task.WhenAll(t1, t2);
    Console.WriteLine(results.Length);
}
catch (Exception ex)
{
    Console.WriteLine("Exception details: {0}", ex);
}

Console.WriteLine();
Console.WriteLine("2. Multiple exceptions with AggregateException");

t1 = GetInfoAsync("Task 1", 3);
t2 = GetInfoAsync("Task 2", 2);
Task<string[]> t3 = Task.WhenAll(t1, t2);
try
{
    string[] results = await t3;
    Console.WriteLine(results.Length);
}
catch
{
    var ae = t3.Exception.Flatten();
    var exceptions = ae.InnerExceptions;
    Console.WriteLine("Exceptions caught: {0}", exceptions.Count);
    foreach (var e in exceptions)
    {
        Console.WriteLine("Exception details: {0}", e);
        Console.WriteLine();
    }
}
```

첫번째 예외만 처리할 수 있다.

잡힌 모든 예외를 처리할 수 있다.

잡아낸 동기화 컨텍스트를 사용해 회피

```
async static void Click(object sender, EventArgs e)
{
    _label.Content = new TextBlock {Text = "Calculating..."};
    TimeSpan resultWithContext = await Test();
    TimeSpan resultNoContext = await TestNoContext();
    //TimeSpan resultNoContext = await TestNoContext().ConfigureAwait(false);
    var sb = new StringBuilder();
    sb.AppendLine(string.Format("With the context: {0}", resultWithContext));
    sb.AppendLine(string.Format("Without the context: {0}", resultNoContext));
    sb.AppendLine(string.Format("Ratio: {0:0.00}",
        resultWithContext.TotalMilliseconds/resultNoContext.TotalMilliseconds));
    _label.Content = new TextBlock {Text = sb.ToString()};
}
```

```
async static Task<TimeSpan> Test()
{
    const int iterationsNumber = 100000;
    var sw = new Stopwatch();
    sw.Start();
    for (int i = 0; i < iterationsNumber; i++)
    {
        var t = Task.Run(() => { });
        await t;
    }
    sw.Stop();
    return sw.Elapsed;
}
```

```
async static Task<TimeSpan> TestNoContext()
{
    const int iterationsNumber = 100000;
    var sw = new Stopwatch();
    sw.Start();
    for (int i = 0; i < iterationsNumber; i++)
    {
        var t = Task.Run(() => { });
        await t.ConfigureAwait(
            continueOnCapturedContext: false);
    }
    sw.Stop();
    return sw.Elapsed;
}
```

사용자 정의 대기 가능 타입 설계

```
static void Main(string[] args)
{
    Task t = AsynchronousProcessing();
    t.Wait();
}

async static Task AsynchronousProcessing()
{
    var sync = new CustomAwaitable(true);
    string result = await sync;
    Console.WriteLine(result);

    var async = new CustomAwaitable(false);
    result = await async;

    Console.WriteLine(result);
}
```

```
class CustomAwaitable
{
    public CustomAwaitable(bool completeSynchronously)
    {
        _completeSynchronously = completeSynchronously;
    }

    public CustomAwaiter GetAwaiter()
    {
        return new CustomAwaiter(_completeSynchronously);
    }

    private readonly bool _completeSynchronously;
}
```

```
class CustomAwaiter : INotifyCompletion
{
    private string _result = "Completed synchronously";
    private readonly bool _completeSynchronously;

    public bool IsCompleted { get { return _completeSynchronously; } }

    public CustomAwaiter(bool completeSynchronously)
    {
        _completeSynchronously = completeSynchronously;
    }

    public string GetResult()
    {
        return _result;
    }

    public void OnCompleted(Action continuation)
    {
        ThreadPool.QueueUserWorkItem( state => {
            Thread.Sleep(TimeSpan.FromSeconds(1));
            _result = GetInfo();
            if (continuation != null) continuation();
        });
    }

    private string GetInfo()
    {
        return string.Format("Task is running on a thread id {0}. Is thread pool thread: {1}",
            Thread.CurrentThread.ManagedThreadId, Thread.CurrentThread.IsThreadPoolThread);
    }
}
```

구문

C# | C++ | F# | VB

```
public interface INotifyCompletion
```

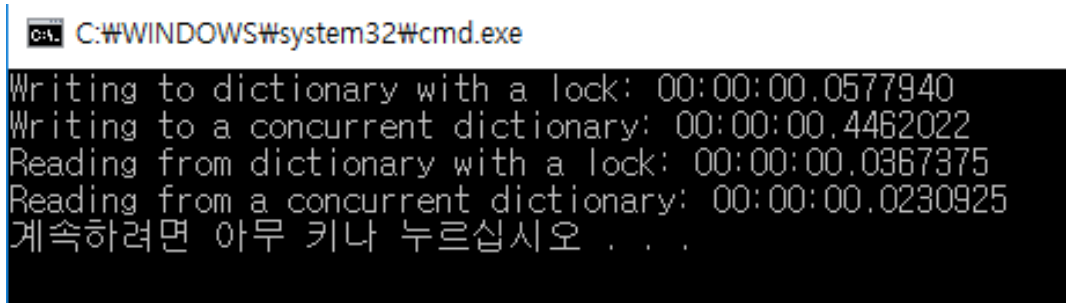
메서드

	이름	설명
	OnCompleted(Action)	인스턴스가 완료 될 때 호출 되는 연속 작업을 예약합니다.

동시성 컬렉션 사용

ConcurrentDictionary 사용

```
var concurrentDictionary = new ConcurrentDictionary<int, string>();
var dictionary = new Dictionary<int, string>();

var sw = new Stopwatch();

sw.Start();
for (int i = 0; i < 1000000; i++)
{
    lock (dictionary)
    {
        dictionary[i] = Item;
    }
}
sw.Stop();
Console.WriteLine("Writing to dictionary with a lock: {0}", sw.Elapsed);

sw.Restart();
for (int i = 0; i < 1000000; i++)
{
    concurrentDictionary[i] = Item;
}
sw.Stop();
Console.WriteLine("Writing to a concurrent dictionary: {0}", sw.Elapsed);

sw.Restart();
for (int i = 0; i < 1000000; i++)
{
    lock (dictionary)
    {
        CurrentItem = dictionary[i];
    }
}
sw.Stop();
Console.WriteLine("Reading from dictionary with a lock: {0}", sw.Elapsed);

sw.Restart();
for (int i = 0; i < 1000000; i++)
{
    CurrentItem = concurrentDictionary[i];
}
sw.Stop();
Console.WriteLine("Reading from a concurrent dictionary: {0}", sw.Elapsed);
```



추가의 경우는 Dictionary에서 lock을 사용하는 것이 더 빠르고, 읽기의 경우는 ConcurrentDictionary가 더 빠르다.

주의할 것은 추가에서 Dictionary가 빠른 경우는 스레드를 1개만 사용했을 때이다. 만약 여러 스레드에서 추가를 하면 ConcurrentDictionary가 빠르다.

ConcurrentQueue를 이용한 비동기 처리 구현

```
static async Task TaskProducer(ConcurrentQueue<CustomTask> queue)
{
    for (int i = 1; i <= 20; i++)
    {
        await Task.Delay(50);
        var workItem = new CustomTask {Id = i};
        queue.Enqueue(workItem);
        Console.WriteLine("Task {0} has been posted", workItem.Id);
    }
}

static async Task TaskProcessor(
    ConcurrentQueue<CustomTask> queue, string name, CancellationToken token)
{
    CustomTask workItem;
    bool dequeueSuccesful = false;

    await GetRandomDelay();
    do
    {
        dequeueSuccesful = queue.TryDequeue(out workItem);
        if (dequeueSuccesful)
        {
            Console.WriteLine("Task {0} has been processed by {1}", workItem.Id, name);
        }

        await GetRandomDelay();
    }
    while (!token.IsCancellationRequested);
}
```

Recipe6-2

ConcurrentStack으로 비동기 처리 순서 변경

```
static async Task TaskProducer(ConcurrentStack<CustomTask> stack)
{
    for (int i = 1; i <= 20; i++)
    {
        await Task.Delay(50);
        var workItem = new CustomTask { Id = i };
        stack.Push(workItem);
        Console.WriteLine("Task {0} has been posted", workItem.Id);
    }
}

static async Task TaskProcessor(
    ConcurrentStack<CustomTask> stack, string name, CancellationToken token)
{
    await GetRandomDelay();
    do
    {
        CustomTask workItem;
        bool popSuccesful = stack.TryPop(out workItem);
        if (popSuccesful)
        {
            Console.WriteLine("Task {0} has been processed by {1}", workItem.Id, name);
        }

        await GetRandomDelay();
    }
    while (!token.IsCancellationRequested);
}
```

ConcurrentBag을 이용해 확장 가능한 크롤러 생성

```
static async Task RunProgram()
{
    var bag = new ConcurrentBag<CrawlingTask>();

    string[] urls = new[] {"http://microsoft.com/", "http://google.com/", "http://facebook.com/", "http://twitter.com/"};

    var crawlers = new Task[4];
    for (int i = 1; i <= 4; i++)
    {
        string crawlerName = "Crawler " + i.ToString();
        bag.Add(new CrawlingTask { UrlToCrawl = urls[i-1], ProducerName = "root"});
        crawlers[i - 1] = Task.Run(() => Crawl(bag, crawlerName));
    }

    await Task.WhenAll(crawlers);
}

static async Task Crawl(ConcurrentBag<CrawlingTask> bag, string crawlerName)
{
    CrawlingTask task;
    while (bag.TryTake(out task))
    {
        IEnumerable<string> urls = await GetLinksFromContent(task);
        if (urls != null)
        {
            foreach (var url in urls)
            {
                var t = new CrawlingTask
                {
                    UrlToCrawl = url,
                    ProducerName = crawlerName
                };

                bag.Add(t);
            }
        }
        Console.WriteLine("Indexing url {0} posted by {1} is completed by {2}!",
            task.UrlToCrawl, task.ProducerName, crawlerName);
    }
}
```

성능이 제일 좋다.
그러나 순서를 보장하지 않는다.

BlockingCollection을 이용한 비동기 처리 일반화

http://blog.naver.com/vactorman/220477582136

내부 Collection에는 lock-free 알고리즘을 사용하는 IProducerConsumerCollection<T>구현체가 들어있다.
(생성 시 전달하면 된다. 아무것도 전달하지 않으면 기본으로 ConcurrentQueue<T>가 사용된다.)

내부 Collection에서 Add / Remove 시 item 한계치에 도달하면 접근한 스레드를 자동으로 Blocking 거는 기능이 들어가 있다.

예를 들어 앞에서 언급한 item이 하나도 없는 상황에서 BlockingCollection<T>의 Take()메서드를 호출하면 내부 Collection에 item이 추가될 때까지 Take()를 호출한 스레드는 Block에 걸리면서 대기하게 된다.

또 반대의 경우에도 사용할 수 있는데 예를 들어 Capcity가 100 인 collection을 내부 Collection으로 지정했다면 현재 item 이 이미 100개에 도달했을 경우 BlockingCollection<T>의 Add()를 호출한 스레드는 내부 컬렉션의 Capcity의 여유가 생기기 전까지 Block이 걸려 대기하게 된다.
(즉, 누군가 하나 이상 item을 빼 가서 Add 가능하게 되어야 block이 풀리면서 Add 하게 된다.)

혹은 꼭 내부 Collection의 Capcity 뿐만 아니라 CompleteAdding() 메서드를 호출하는 것을 통해 현재까지 Add된 item까지만 처리하도록 할 수 있는데(즉, 더 이상 item을 add 하지 못하게 막게 된다.)

이 때 Add 를 시도하면 스레드가 Block 되는 것이 아니라 **예외가 발생**하게 된다.

그리고 이 CompleteAdding() 메서드 호출 이후에는 내부 Collection이 비어 있어도 Take()호출 시 스레드가 Blocking이 걸리지 않고 마찬가지로 예외를 뚜드러맞게 된다.

CompleteAdding() 이 메서드는 거의 내부 Collection을 봉인할 때나 사용할만한 메서드이니 조심해서 써야한다. (한 번 호출하고 나면 저 상태를 풀 방법도 없다.)

IsAddingCompleted
BlockingCollection<T>가 추가 완료(CompleteAdding())로 되어 있는지 확인한다.

IsCompleted
BlockingCollection<T>가 추가 완료로 되어 있고 비었는지 확인한다.

```
static void Main(string[] args)
{
    Console.WriteLine("Using a Queue inside of BlockingCollection");
    Console.WriteLine();
    Task t = RunProgram();
    t.Wait();

    Console.WriteLine();
    Console.WriteLine("Using a Stack inside of BlockingCollection");
    Console.WriteLine();
    t = RunProgram(new ConcurrentStack<CustomTask>());
    t.Wait();
}
```

```
static async Task RunProgram(IProducerConsumerCollection<CustomTask> collection = null)
{
    var taskCollection = new BlockingCollection<CustomTask>();
    if(collection != null)
        taskCollection= new BlockingCollection<CustomTask>(collection);

    var taskSource = Task.Run(() => TaskProducer(taskCollection));

    Task[] processors = new Task[4];
    for (int i = 1; i <= 4; i++)
    {
        string processorId = "Processor " + i;
        processors[i - 1] = Task.Run(
            () => TaskProcessor(taskCollection, processorId));
    }

    await taskSource;

    await Task.WhenAll(processors);
}
```

```
static async Task TaskProducer(BlockingCollection<CustomTask> collection)
{
    for (int i = 1; i <= 20; i++)
    {
        await Task.Delay(20);
        var workItem = new CustomTask { Id = i };
        collection.Add(workItem);
        Console.WriteLine("Task {0} has been posted", workItem.Id);
    }
    collection.CompleteAdding();
}
```

비동기 I/O 사용

비동기 HTTP 서버와 클라이언트 작성

```
using (var client = new HttpClient())
{
    HttpResponseMessage responseMessage = await client.GetAsync(url);
    string responseHeaders = responseMessage.Headers.ToString();
    string response = await responseMessage.Content.ReadAsStringAsync();

    Console.WriteLine("Response headers:");
    Console.WriteLine(responseHeaders);
    Console.WriteLine("Response body:");
    Console.WriteLine(response);
}
```

```
readonly HttpListener _listener;
const string RESPONSE_TEMPLATE = "<html><head><title>Test</title></head><

public AsyncHttpServer(int portNumber)
{
    _listener = new HttpListener();
    _listener.Prefixes.Add(string.Format("http://+:{0}/", portNumber));
}

public async Task Start()
{
    _listener.Start();

    while (true)
    {
        var ctx = await _listener.GetContextAsync();
        Console.WriteLine("Client connected...");
        var response = string.Format(RESPONSE_TEMPLATE, DateTime.Now);

        using (var sw = new StreamWriter(ctx.Response.OutputStream))
        {
            await sw.WriteAsync(response);
            await sw.FlushAsync();
        }
    }
}
```

비동기적으로 데이터베이스 작업

```
using (var connection = new SqlConnection(dbConnectionString))
{
    await connection.OpenAsync();

    var cmd = new SqlCommand("SELECT newid()", connection);
    var result = await cmd.ExecuteScalarAsync();

    Console.WriteLine("New GUID from DataBase: {0}", result);

    cmd = new SqlCommand(@"CREATE TABLE [dbo].[CustomTable]( [ID] [int] IDENTITY(1,
_ID] PRIMARY KEY CLUSTERED ([ID] ASC) ON [PRIMARY]) ON [PRIMARY]", connection);
    await cmd.ExecuteNonQueryAsync();

    Console.WriteLine("Table was created succesfully.");

    cmd = new SqlCommand(@"INSERT INTO [dbo].[CustomTable] (Name) VALUES ('John');
o].[CustomTable] (Name) VALUES ('Peter');
o].[CustomTable] (Name) VALUES ('James');
o].[CustomTable] (Name) VALUES ('Eugene');", connection);
    await cmd.ExecuteNonQueryAsync();

    Console.WriteLine("Inserted data succesfully");
    Console.WriteLine("Reading data from table...");

    cmd = new SqlCommand(@"SELECT * FROM [dbo].[CustomTable]", connection);
    using (SqlDataReader reader = await cmd.ExecuteReaderAsync())
    {
        while (await reader.ReadAsync())
        {
            var id = reader.GetFieldValue<int>(0);
            var name = reader.GetFieldValue<string>(1);

            Console.WriteLine("Table row: Id {0}, Name {1}", id, name);
        }
    }
}
```

Recipe9-3

병렬 프로그래밍 패턴

TPL 데이터플로우로 병렬 파이프라인 구현

```
var inputBlock = new BufferBlock<int>(
    new DataflowBlockOptions { BoundedCapacity = 5, CancellationToken = cts.Token });

var filter1Block = new TransformBlock<int, decimal>(
    n =>
    {
        decimal result = Convert.ToDecimal(n * 0.97);
        Console.WriteLine("Filter 1 sent {0} to the next stage on thread id {1}", result,
            Thread.CurrentThread.ManagedThreadId);
        Thread.Sleep(TimeSpan.FromMilliseconds(100));
        return result;
    }
    , new ExecutionDataflowBlockOptions { MaxDegreeOfParallelism = 4, CancellationToken = cts.Token });

var filter2Block = new TransformBlock<decimal, string>(
    n =>
    {
        string result = string.Format("--{0}--", n);
        Console.WriteLine("Filter 2 sent {0} to the next stage on thread id {1}", result,
            Thread.CurrentThread.ManagedThreadId);
        Thread.Sleep(TimeSpan.FromMilliseconds(100));
        return result;
    }
    , new ExecutionDataflowBlockOptions { MaxDegreeOfParallelism = 4, CancellationToken = cts.Token });

var outputBlock = new ActionBlock<string>(
    s =>
    {
        Console.WriteLine("The final result is {0} on thread id {1}",
            s, Thread.CurrentThread.ManagedThreadId);
    }
    , new ExecutionDataflowBlockOptions { MaxDegreeOfParallelism = 4, CancellationToken = cts.Token });
```

이 블록 안에 5개의 항목이 있으면 하나라도 처리 되기전까지는 블럭된다.

데이터 변환을 위한 블럭.

작업 스레드의 동시 최대 값

모든 들어오는 항목에 특정 동작을 실행한다.

블럭을 연결한다

단계가 끝났을 때 자동으로 결과 예외를 다음 단계로 전파한다.

```
inputBlock.LinkTo(filter1Block, new DataflowLinkOptions { PropagateCompletion = true });
filter1Block.LinkTo(filter2Block, new DataflowLinkOptions { PropagateCompletion = true });
filter2Block.LinkTo(outputBlock, new DataflowLinkOptions { PropagateCompletion = true });

try
{
    Parallel.For(0, 20, new ParallelOptions { MaxDegreeOfParallelism = 4, CancellationToken = cts.Token }
    , i =>
    {
        Console.WriteLine("added {0} to source data on thread id {1}", i, Thread.CurrentThread.ManagedThreadId);
        inputBlock.SendAsync(i).GetAwaiter().GetResult();
    });
    inputBlock.Complete();
    await outputBlock.Completion;
    Console.WriteLine("Press ENTER to exit.");
}
catch (OperationCanceledException)
{
    Console.WriteLine("Operation has been canceled! Press ENTER to exit.");
}

Console.ReadLine();
```

BufferBlock<T>.Complete()
이 이상 메시지를 받지 않거나 또는 생성시키지 않는다. 지연 메시지를 사용하지 않음을 IDataflowBlock에 통지한다.

```
Parallel.For(0, 20, new ParallelOptions { MaxDegreeOfPa
, i =>
{
    Console.WriteLine("added {0} to source data on thre
    inputBlock.SendAsync(i).GetAwaiter().GetResult();
});
inputBlock.Complete();
Console.WriteLine("inputBlock.Complete() 호출");

await outputBlock.Completion;
Console.WriteLine("Press ENTER to exit.");
```

```
Filter 2 sent --18.43-- to the next stage on threa
The final result is --16.49-- on thread id 11
The final result is --11.64-- on thread id 4
The final result is --6.79-- on thread id 8
The final result is --17.46-- on thread id 10
The final result is --18.43-- on thread id 6
Press ENTER to exit.
```

```
Parallel.For(0, 20, new ParallelOptions { MaxDegreeOfPar
, i =>
{
    Console.WriteLine("added {0} to source data on threa
    inputBlock.SendAsync(i).GetAwaiter().GetResult();
});
//inputBlock.Complete();
//Console.WriteLine("inputBlock.Complete() 호출");

await outputBlock.Completion;
Console.WriteLine("Press ENTER to exit.");
```

```
The final result is --18.43-- on thread id 8
Filter 2 sent --8.73-- to the next stage on thread id 4
The final result is --7.76-- on thread id 7
Filter 2 sent --3.88-- to the next stage on thread id 5
The final result is --2.91-- on thread id 8
The final result is --16.49-- on thread id 6
The final result is --6.79-- on thread id 8
The final result is --8.73-- on thread id 4
The final result is --3.88-- on thread id 5
```