[목차]

1. 쓰레드(Thread) 2

2. 스레드 동기화 17

3. 비동기 호출 28

1. 쓰레드(Thread)

명령어를 실행하기 위한 스케줄링 단위이며, 프로세스 내부에서 생성할 수 있다. 하나의 프로세스는 여러 개의 스레드 자원을 가질 수 있다.

윈도우는 프로세스를 생성할 때 기본적으로 한 개의 스레드를 생성하며, 이를 주 스레드(main thread, primary thread)라고 한다.

스레드는 CPU 명령어 실행과 관련된 정보를 보관하고 있는데, 이를 스레드 문맥(thread context)라 한다. 운영체제이 스케줄러는 실행 되야 할 적절한 스레드를 골라서 CPU로 하여금 실행되게 하며, 이 때 두 가지 동작을 수행한다. CPU는 현재 실행 중인 스레드를 다음에 다시 이어서 실행할 수 있게 CPU의 환경 정보를 스레드 문맥에 보관한다. 그리고 운영체제로부터 할당 받은 스레드의 문맥 정보를 다시 CPU 내부로 로드 해서 마치 해당 스레드가 실행되고 있었던 상태인 것처럼 복원한 다음, 일정 시간 동안 실행을 계속한다.

1. 스레드 상태 정보

```
using System;
using System.Threading;

class Program
{
    static void Main(string[] args)
    {
        Thread thread = Thread.CurrentThread;
        Console.WriteLine(thread.ThreadState); // 출력 결과: Running

        Console.WriteLine(DateTime.Now); // 출력 결과: 2013-02-17 오후 11:02:33
        Thread.Sleep(1000); // 1초 동안 스레드 중지
        Console.WriteLine(DateTime.Now); // 출력 결과: 2013-02-17 오후 11:02:34
    }
}
```

* Sleep 메서드는 ThreadState.WaitSleepJoin 상태로 변경한다.

예) 스레드 추가 정보

```
using System;
using System.Diagnostics;

public class ThreadInfo
{
    public static void Main()
    {
        Process proc = Process.GetCurrentProcess();      // 현재 프로세스 정보 가져오기
        ProcessThreadCollection ths = proc.Threads;      // 스레드 정보 가져오기

        int threadID;                                     // 스레드 ID 번호
        DateTime startTime;                               // 스레드 시작 시간
        int priority;                                     // 스레드 우선순위
        ThreadState thstate;                              // 스레드 상태

        int index = 1;                                    // 스레드 번호 출력

        Console.WriteLine(" 현재 프로세스에서 실행중인 스레드 수: " + ths.Count);

        foreach (ProcessThread pth in ths)
        {
            threadID = pth.Id;
            startTime = pth.StartTime;
            priority = pth.BasePriority;
            thstate = pth.ThreadState;

            Console.WriteLine("******* {0} 스레드 정보 *****", index++);
            Console.WriteLine(" ID: {0}₩n 시작시간: {1}₩n Priority: {2}₩n 스레드 상태:{3}₩n",
                    threadID, startTime, priority, thstate);
        }
    }
}
```

2. 스레드 생성 및 실행

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

	static void threadFunc()	{
		Console.WriteLine("threadFunc run!");
	}
}
```

3. 2개의 스레드 실행이 완료된 후 프로그램 종료

```
using System;
using System.Threading;
class Program{
	static void Main(string[] args)	{
		Thread t = new Thread(threadFunc);
		// t.IsBackground = true;	// 기본은 foreground thread
		t.Start();
		// 더는 주 스레드가 실행할 명령어가 없으므로 주 스레드는 제거됨
	}

	static void threadFunc()	{
		Console.WriteLine("60초 후에 프로그램 종료");
		Thread.Sleep(1000 * 60); // 60초 동안 실행 중지
					// 현재 주 스레드는 종료됐어도 t 스레드는 존속한다.
		Console.WriteLine("스레드 종료!");
	}
}
```

* 프로그램의 실행 종료에 영향을 미치는 스레드를 가리켜 전경 스레드(foreground thread)라 한다. 배경 스레드(background thread)도 있으며, 이 유형은 실행 종료에 영향을 미치지 않는다.

4. Join 메서드 사용 예

```
using System;
using System.Threading;
class Program{
	static void Main(string[] args)	{
		Thread t = new Thread(threadFunc);
		t.IsBackground = true;
		t.Start();

		t.Join(); // t 스레드가 종료될 때까지 현재 스레드를 무한 대기
		Console.WriteLine("주 스레드 종료!");
	}

	static void threadFunc()	{
		Console.WriteLine("60초 후에 프로그램 종료");
		Thread.Sleep(1000 * 60); // 60초 동안 실행 중지
		Console.WriteLine("스레드 종료!");
	}
}
```

* Join 메서드는 background thread가 종료될 때까지 대기

5. 스레드 상태 제어

```
/*************************************************************
 * 스레드 상태 제어
 *  - Abort()        : 스레드 강제 종료(스레드 종료 방법)
 *  - Interrupt()    : 대기 스레드 상태에 있는 스레드 중단
 *  - Join()         : 스레드가 종료될 때까지 호출 스레드 중지
 *  - Resume()       : 일시 중지된 스레드를 다시 실행
 *  - Start()        : 현재 스레드 상태를 Running으로 변경
 *  - Stop()         : 현재 스레드를 Stopped 시킴(시스템에 문제 발생)
 *  - Suspend()      : 스레드의 실행을 일시 중단
 *  - ToStrign()     : 스레드 객체를 문자열로 봔한
*************************************************************/
using System;
using System.Threading;

public class ThreadExam4
{
    public static void Print()     // 스레드로 실행될 메서드
    {
        try
        {
            for (int i = 0; i < 1000; i++)
            {
                Console.WriteLine(" Print 스레드 : {0} ", i);
            }
        }
        catch (ThreadAbortException ex)
        {
            Console.WriteLine("스레드 에러: " + ex.Message);
        }
    }

    public static void TInfo(Thread th)
    {
        Console.WriteLine("Thread ID : {0} ₩t 상태 : {1}", th.GetHashCode(), th.ThreadState);
    }
```

```
    public static void Main()
    {
        string msg = null;    // 스레드 정보를 출력할 문자열

        Thread th = new Thread(new ThreadStart(Print));
        TInfo(th);      // 스레드 정보 출력

        th.Start();              // 스레드 시작 (Stated 상태로 변화됨)
        Thread.Sleep(100);   // th 스레드가 시작될 때까지 잠시 대기함
        TInfo(th);               // 스레드 상태가 UnStarted에서 Started로 변경

        th.Suspend();            // 스레드 Suspend(일시정지)
        Thread.Sleep(100);   // 스레드가 일시 정지될 시간적 여유를 줌

        // 콘솔 화면은 Print 스레드가 사용하기 때문에 MessageBox로 화면에 스레드 정보 출력
        msg = "Thread ID :" + th.GetHashCode() + "₩t 상태 : " + th.ThreadState.ToString();
        System.Windows.Forms.MessageBox.Show(msg);

        th.Resume();             // Suspend 상태 스레드를 깨움
        Thread.Sleep(100);       // Resume 이 적용될 시간적 여유를 줌

        msg = "Thread ID :" + th.GetHashCode() + "₩t 상태 : " + th.ThreadState.ToString();
        System.Windows.Forms.MessageBox.Show(msg);

        th.Abort();              // 스레드 종료 메시지
        th.Join();               // 스레드가 완전히 정지될 때까지 기다림

        TInfo(th);                   // 스레드 상태 정보 출력
    }
}
```

6. 스레드 우선 순위

```
/*****************************************************************
 * 스레드 우선 순위
 *    ThreadPriority.Normal : 기본 우선 순위
 *    ThreadPriority.BelowNormal,
 *    ThreadProirity.Lowest : 우선순위 가장 낮음
    1. 스레드 우선순위를 지정하지 않은 상태 : 모두 Normal상태
 *       Start() 호출한 순서대로 스레드가 실행됨
    2. 스레드 우선 순위 부여
 *****************************************************************/
using System;
using System.Threading;

//1
public class ThreadExam5
{
    public static void Print1() { Console.WriteLine("Print1 스레드 *"); }
    public static void Print2() { Console.WriteLine("Print2 스레드 **"); }
    public static void Print3() { Console.WriteLine("Print3 스레드 ***"); }
    public static void Print4() { Console.WriteLine("Print4 스레드 ****"); }
    public static void Print5() { Console.WriteLine("Print5 스레드 *****"); }

    public static void Main()
    {
        // 스레드 개체 생성
        Thread th1 = new Thread(new ThreadStart(Print1));
        Thread th2 = new Thread(new ThreadStart(Print2));
        Thread th3 = new Thread(new ThreadStart(Print3));
        Thread th4 = new Thread(new ThreadStart(Print4));
        Thread th5 = new Thread(new ThreadStart(Print5));

        th2.Start();          // 스레드 실행
        th5.Start();
        th3.Start();
        th4.Start();
        th1.Start();
    }
```

```
}

/*
//2
public class ThreadExam6
{
    public static void Print1() { Console.WriteLine("Print1 스레드 *"); }
    public static void Print2() { Console.WriteLine("Print2 스레드 **"); }
    public static void Print3() { Console.WriteLine("Print3 스레드 ***"); }
    public static void Print4() { Console.WriteLine("Print4 스레드 ****"); }
    public static void Print5() { Console.WriteLine("Print5 스레드 *****"); }

    public static void Main()
    {
        Thread th1 = new Thread(new ThreadStart(Print1));
        Thread th2 = new Thread(new ThreadStart(Print2));
        Thread th3 = new Thread(new ThreadStart(Print3));
        Thread th4 = new Thread(new ThreadStart(Print4));
        Thread th5 = new Thread(new ThreadStart(Print5));

        th1.Priority = ThreadPriority.Highest;    // th1 스레드 우선순위 부여
        th5.Priority = ThreadPriority.Lowest;     // th5 스레드 우선순위 부여

        th2.Start();          // 스레드 시작
        th5.Start();          // 스레드 시작 순위가 밀림
        th3.Start();
        th4.Start();
        th1.Start();
    }
}
*/
```

7. 스레드 콜백 처리

```
using System;
using System.Threading;
public class CallbackExam {
    private ExampleCallback callback;

    public CallbackExam(ExampleCallback callbackDelegate)    {
        callback = callbackDelegate;
    }
    public void ThreadProc()    {
        Console.WriteLine(Thread.CurrentThread.Name + " : ThreadProc() 호출");
        Thread.Sleep(1000);
        if (callback != null)
            callback(1000);
    }
}
public delegate void ExampleCallback(int lineCount);

public class Example{
    private static int examdata = 0;

    public static void ResultCallback(int data)    {
        Console.WriteLine(Thread.CurrentThread.Name + " 스레드가 실행한 ResultCallback : " + data);
        Example.examdata = data;
    }
    public static void ShowData()    {
        Console.WriteLine("Example.ShowData() => " + Example.examdata);
    }
    public static void Main()    {
        CallbackExam ce = new CallbackExam(new ExampleCallback(ResultCallback)
       );
        Thread th = new Thread(new ThreadStart(ce.ThreadProc));
        th.Name = "Thread A";
        th.Start();
        Console.WriteLine("th 스레드 Start() 메서드 실행 후~");
        th.Join();
        Example.ShowData();
```

```
    }
}
```

[스레드에서 콜백을 이용한 데이터 처리]

1) 스레드 실행시 콜백 메서드 등록

2) 콜백은 스레드 간 데이터를 교환하는 주요 방법임

- A 스레드에서 B 스레드를 생성할 때,
  B 스레드 콜백을 A 스레드에 등록해 두면
  B 스레드에서 특정 이벤트가 발생할 때
  A 스레드에 등록한 콜백을 통해 A스레드에 값을 전달 함

8. 스레드 인자 전달(1)

```
using System;
using System.Threading;
class Program{
	static void Main(string[] args)	{
		Thread t = new Thread(threadFunc);
		t.Start(10);
	}
	static void threadFunc(object initialValue)	{
		int intValue = (int)initialValue;
		Console.WriteLine(intValue); // 출력 결과: 10
	}
}
```

9. 스레드 인자 전달(2)

```
using System;
using System.Threading;
class ThreadParam {
	public int Value1;
	public int Value2;
}
class Program{
	static void Main(string[] args)	{
		Thread t = new Thread(threadFunc);

		ThreadParam param = new ThreadParam();
		param.Value1 = 10;
		param.Value2 = 20;
		t.Start(param);
	}

	static void threadFunc(object initialValue)	{
		ThreadParam value = (ThreadParam)initialValue;
		Console.WriteLine("{0}, {1}", value.Value1, value.Value2); // 출력 결과: 10, 20
	}
}
```

10. 스레드를 사용하지 않는 계산 프로그램

```
using System;
class Program{
    static void Main(string[] args)    {
        Console.WriteLine("입력한 숫자까지의 소수 개수 출력 (종료: 'x' + Enter)");
        while (true)        {
            Console.WriteLine("숫자를 입력하세요.");
            string userNumber = Console.ReadLine();
            if (userNumber.Equals("x", StringComparison.OrdinalIgnoreCase) == true)        {
                Console.WriteLine("프로그램 종료!");
                break;
            }
            CountPrimeNumbers(userNumber);
        }
    }
    static void CountPrimeNumbers(object initialValue)    {
        string value = (string)initialValue;
        int primeCandidate = int.Parse(value);
        int totalPrimes = 0;
        for (int i = 2; i < primeCandidate; i++)        {
            if (IsPrime(i) == true)        {
                totalPrimes++;
            }
        }
        Console.WriteLine("숫자 {0}까지의 소수 개수? {1}", value, totalPrimes);
    }
    static bool IsPrime(int candidate)    {    // 소수 판정 메서드
        if ((candidate & 1) == 0)        {
            return candidate == 2;
        }
        for (int i = 3; (i * i) <= candidate; i += 2)        {
            if ((candidate % i) == 0) return false;
        }
        return candidate != 1;
    }
}
```

* 100000000 정도 되면 주 스레드가 계산 작업이 수행되는 동안 사용자는 다른 어떤 키도 입력이 불가능.

11. 스레드를 사용한 계산기 프로그램

```
using System;
using System.Threading;
class Program{
    static void Main(string[] args)    {
        Console.WriteLine("입력한 숫자까지의 소수 개수 출력 (종료: 'x' + Enter)");
        while (true)          {
            Console.WriteLine("숫자를 입력하세요.");
            string userNumber = Console.ReadLine();
            if (userNumber.Equals("x", StringComparison.OrdinalIgnoreCase) == true)                {
                Console.WriteLine("프로그램 종료!");
                break;
            }
            Thread t = new Thread(CountPrimeNumbers);
            t.IsBackground = true;
            t.Start(userNumber);
        }
    }
    static void CountPrimeNumbers(object initialValue)    {
        string value = (string)initialValue;
        int primeCandidate = int.Parse(value);
        int totalPrimes = 0;
        for (int i = 2; i < primeCandidate; i++)          {
            if (IsPrime(i) == true)
                totalPrimes++;
        }
        Console.WriteLine("숫자 {0}까지의 소수 개수? {1}", value, totalPrimes);
    }
.    static bool IsPrime(int candidate)    {
        if ((candidate & 1) == 0)
            return candidate == 2;
        for (int i = 3; (i * i) <= candidate; i += 2)
            if ((candidate % i) == 0) return false;
        return candidate != 1;
    }
}
```

* 1000000000을 입력한 후, x+Enter를 입력하면 프로그램이 종료된다.

12. ThreadPool

스레드 동작 방식은 Thread 타입의 생성자에 전달되는 메서드의 코드 유형에 따라 두 가지로 나뉜다.

1) 상시 실행

   스레드가 일단 생성되면 비교적 오랜 시간 동안 생성돼 있는 유형

   ex) 특정 디렉토리의 변화를 감시

2) 1회성의 임시 실행

   특정 연산만을 수행하고 바로 종료

2번의 경우 매번 스레드를 생성하는 것은 비효율적이다. 임시적인 목적으로 언제든 원하는 때에 스레드를 사용할 수 있다면 좋을 것이다. CLR은 이 같은 용도로 사용할 수 있는 기본적인 스레드 풀을 마련해 두었다.

스레드 풀은 "재사용할 수 있는 자원의 집합"을 의미한다. 따라서 스레드 풀이라 하면 필요할 때마다 스레드를 꺼내 쓰고 필요 없어지면 다시 풀에 스레드가 반환되는 기능을 일컫는다.

```
using System;
using System.Threading;
public class ThreadExam2 {
    static int i = 0;
    // 첫 번째 스레드
    public static void Print1(object obj)    {
        for (i = 0; i < 3; i++)
        {
            Console.WriteLine("첫 번째 Thread :{0} ***", i);
            Thread.Sleep(100);    // 0.1 초 동안 스레드 정지
        }
    }
    // 두 번째 스레드
    public void Print2(object obj)    {
        for (int i = 0; i < 3; i++)        {
            Console.WriteLine("두 번째 Thread :{0} ***", i);
            Thread.Sleep(100);    // 0.1 초 동안 스레드 정지
        }
     }
```

```
    public static void Main()      {
        // static method를 이용한 ThreadPool 에 대한 작업 요청
        //첫 번째 인자 : 스레드 함수
        //두 번째 인자 : 전달 인자
        ThreadPool.QueueUserWorkItem(new WaitCallback(Print1), null);

        // instance method를 이용한 ThreadPool 에 대한 작업 요청
        ThreadPool.QueueUserWorkItem(new WaitCallback((new ThreadExam2()).Print2), null);

        for (int i = 0; i < 10; i++)            {
            Console.WriteLine("main: {0}", i);
            Thread.Sleep(100);   // 메인 스레드
        }
    }
}
```

2. 스레드 동기화

다중 스레드를 사용할 때 동기화 문제 발생

1. 다중 스레드에서 단일 변수 사용(공유 리소스 문제)

```
using System;
using System.Threading;

class Program{
    int number = 0;

    static void Main(string[] args)    {
        Program pg = new Program();

        Thread t1 = new Thread(threadFunc);
        Thread t2 = new Thread(threadFunc);

        t1.Start(pg);
        t2.Start(pg); // 2개의 스레드를 시작하고,

        t1.Join();
        t2.Join(); // 2개의 스레드 실행이 끝날 때까지 대기한다.

        Console.WriteLine(pg.number); // 스레드 실행 완료 후 number 필드 값을 출력
    }

    static void threadFunc(object inst)    {
        Program pg = inst as Program;

        for (int i = 0; i < 10; i++)
        // for (int i = 0; i < 10000; i++)
        {
            pg.number = pg.number + 1; // Program 객체의 number 필드 값을 증가
        }
    }
}
```

* 예측할 수 없는 값이 나옴.

2. Monotor를 이용한 동기화

```
using System;
using System.Threading;
class Program{
    int number = 0;
    static void Main(string[] args)    {
        Program pg = new Program();

        Thread t1 = new Thread(threadFunc);
        Thread t2 = new Thread(threadFunc);

        t1.Start(pg); //동일한 객체가 전달됨
        t2.Start(pg); // 2개의 스레드를 시작하고,

        t1.Join();
        t2.Join(); // 2개의 스레드 실행이 끝날 때까지 대기한다.

        Console.WriteLine(pg.number); // 스레드 실행 완료 후 number 필드 값을 출력
    }

    static void threadFunc(object inst)    {
        Program pg = inst as Program;
        for (int i = 0; i < 100000; i++)
        {
            Monitor.Enter(pg);
             try
             {
                 pg.number = pg.number + 1;
             }
             finally
             {
                 Monitor.Exit(pg);
             }
       }
    }
}
```

3. lock을 이용한 동기화

```
using System;
using System.Threading;

class Program
{
    int number = 0;

    static void Main(string[] args)
    {
        Program pg = new Program();

        Thread t1 = new Thread(threadFunc);
        Thread t2 = new Thread(threadFunc);

        t1.Start(pg);
        t2.Start(pg); // 2개의 스레드를 시작하고,

        t1.Join();
        t2.Join(); // 2개의 스레드 실행이 끝날 때까지 대기한다.

        Console.WriteLine(pg.number); // 스레드 실행 완료 후 number 필드 값을 출력
    }

    static void threadFunc(object inst)     {
        Program pg = inst as Program;
        for (int i = 0; i < 100000; i++)
        {
            lock (pg)
            {
                pg.number = pg.number + 1;
            }
        }
    }
}
```

* C# 컴파일러에 의해 try/finally + Monitor.Enter / Exit 코드로 변경된다.

* 코드가 간결 하기 때문에 lock 예약어를 이용하는 구문이 더 선호된다.

4. 스레드에 안전하지 않은 메서드

```
using System;
using System.Threading;

class MyData{
    int number = 0;

    public int Number { get { return number; } }

    public void Increment()    {
        number++;
    }
}

class Program{
    static void Main(string[] args)    {
        MyData data = new MyData();

        Thread t1 = new Thread(threadFunc);
        Thread t2 = new Thread(threadFunc);

        t1.Start(data);
        t2.Start(data);

        t1.Join();
        t2.Join();

        Console.WriteLine(data.Number);
    }

    static void threadFunc(object inst)    {
        MyData data = inst as MyData;
        for (int i = 0; i < 100000; i++)        {
            data.Increment();
        }
    }
}
```

5. 스레드에 안전한 메서드

```
using System;
using System.Threading;
class MyData{
    int number = 0;
    public object _numberLock = new object();
    public int Number { get { return number; } }

    public void Increment()    {
        lock (_numberLock)
        {
            number++;
        }
    }
}
class Program{
    static void Main(string[] args)    {
        MyData data = new MyData();

        Thread t1 = new Thread(threadFunc);
        Thread t2 = new Thread(threadFunc);

        t1.Start(data);  //동일한 객체 전달
        t2.Start(data);  //동일한 객체 전달

        t1.Join();
        t2.Join();
        Console.WriteLine(data.Number);
    }
    static void threadFunc(object inst)    {
        MyData data = inst as MyData;

        for (int i = 0; i < 100000; i++)        {
            data.Increment();
        }
    }
}
```

6. 스레드에 안전하지 않은 메서드를 외부에서 안전하게 사용하는 방법

```
using System;
using System.Threading;
class MyData {
    int number = 0;
    public int Number { get { return number; } }
    public void Increment()    {
        number++;
    }
}

class Program{
    static void Main(string[] args)    {
        MyData data = new MyData();

        Thread t1 = new Thread(threadFunc);
        Thread t2 = new Thread(threadFunc);

        t1.Start(data);
        t2.Start(data);

        t1.Join();
        t2.Join();

        Console.WriteLine(data.Number);
    }
    static void threadFunc(object inst)    {
        MyData data = inst as MyData;

        for (int i = 0; i < 100000; i++)        {
            lock (data)
            {
                data.Increment();
            }
        }
    }
}
```

7. System.Threading.Interlocked

Interlocked 클래스는 정적 클래스이다. 다중 스레드 환경에서 공유 자원을 간단하게 동기화 할 수 있다.

```
using System;
using System.Threading;
class Program{
    static void Main(string[] args)    {
        MyData data = new MyData();

        Thread t1 = new Thread(threadFunc);
        Thread t2 = new Thread(threadFunc);

        t1.Start(data);
        t2.Start(data);

        t1.Join();
        t2.Join();

        Console.WriteLine(data.Number);
    }
    static void threadFunc(object inst)    {
        MyData data = inst as MyData;

        for (int i = 0; i < 100000; i++)        {
            data.Increment();
        }
    }
}
class MyData{
    int number = 0;

    public int Number { get { return number; } }

    public void Increment()    {
        Interlocked.Increment(ref number);
    }
}
```

8. System.Threading.ThreadPool

```
using System;
using System.Threading;

class MyData{
    int number = 0;
    public int Number { get { return number; } }
    public void Increment()    {
        Interlocked.Increment(ref number);
    }
}

class Program{
    static void Main(string[] args)    {
        MyData data = new MyData();

        ThreadPool.QueueUserWorkItem(threadFunc, data);
        ThreadPool.QueueUserWorkItem(threadFunc, data);

        Thread.Sleep(1000);

        Console.WriteLine(data.Number);
    }

    static void threadFunc(object inst)    {
        MyData data = inst as MyData;

        for (int i = 0; i < 100000; i++)
        {
            data.Increment();
        }
    }
}
```

* ThreadPool은 프로그램 시작과 동시에 0개의 스레드를 가지며 생성된다.

* 첫번째 QueueUserWorkItem 을 호출했을 때 ThreadPool에 자동으로 1개의 스레드를 생성해 thread Func을 할당해 실행한다.

9. System.Threading.EventWaitHandle

EventWaitHandle은 Monotor 타입처럼 스레드 동기화 수단의 하나이다. 스레드로 하여금 이벤트를 기다리게 만들 수 있고, 다른 스레드에서는 원하는 이벤트를 발생시키는 시나리오에 적합하다. 이벤트 객체는 Signal, non-Signal 상태를 갖고, 상태변화는 Set, ReSet 메서드로 전환된다.

이와 함께 이벤트 객체는 WaitOne 메서드를 제공한다. 어떤 스레드가 WaitOne 메서드를 호출하는 시점에 이벤트 객체가 Signal 상태이면 메서드에서 곧바로 제어가 반환되지만, Non-Signal 상태였다면 이벤트 객체가 Signal 상태로 바뀔 때까지 WaitOne 메서드는 제어를 반환하지 않는다.

```
using System;
using System.Threading;
class Program{
    static void Main(string[] args)    {
        // Non-Signal 상태의 이벤트 객체 생성
        // 생성자의 첫 번째 인자가 false이면 Non-Signal 상태로 시작.
        // true이면 Signal 상태로 시작
        EventWaitHandle ewh = new EventWaitHandle(false, EventResetMode.ManualReset);

        Thread t = new Thread(threadFunc);
        t.IsBackground = true;
        t.Start(ewh);

        // Non-Signal 상태에서 WaitOne을 호출했으므로 Signal 상태로 바뀔 때까지 대기
        ewh.WaitOne();

        Console.WriteLine("주 스레드 종료!");
    }
    static void threadFunc(object state)    {
        EventWaitHandle ewh = state as EventWaitHandle;

        Console.WriteLine("5초 후에 프로그램 종료");
        Thread.Sleep(1000 * 5); // 5초 동안 실행 중지
        Console.WriteLine("스레드 종료!");

        // Non-Signal 상태의 이벤트를 Signal 상태로 전환
        ewh.Set();
    }
}
```

10. 개선된 ThreadPool의 사용 예

```
using System;
using System.Collections;
using System.Threading;
class MyData{
	int number = 0;
	public int Number { get { return number; } }
	public void Increment()	{
		Interlocked.Increment(ref number);
	}
}
class Program{
	static void Main(string[] args)	{
		MyData data = new MyData();

		Hashtable ht1 = new Hashtable();
		ht1["data"] = data;
		ht1["evt"] = new EventWaitHandle(false, EventResetMode.ManualReset);

		// 데이터와 함께 이벤트 객체를 스레드 풀의 스레드에 전달한다.
		ThreadPool.QueueUserWorkItem(threadFunc, ht1);
		Hashtable ht2 = new Hashtable();

		ht2["data"] = data;
		ht2["evt"] = new EventWaitHandle(false, EventResetMode.ManualReset);

		// 데이터와 함께 이벤트 객체를 스레드 풀의 스레드에 전달한다.
		ThreadPool.QueueUserWorkItem(threadFunc, ht2);

		// 2개의 이벤트 객체가 Signal 상태로 바뀔 때까지 대기한다.
		(ht1["evt"] as EventWaitHandle).WaitOne();
		(ht2["evt"] as EventWaitHandle).WaitOne();

		Console.WriteLine(data.Number);
	}
```

```
    static void threadFunc(object inst)    {
        Hashtable ht = inst as Hashtable;
        MyData data = ht["data"] as MyData;

        for (int i = 0; i < 100000; i++)
        {
            data.Increment();
        }

        // 주어진 이벤트 객체를 Signal 상태로 전환한다.
        (ht["evt"] as EventWaitHandle).Set();
    }
}
```

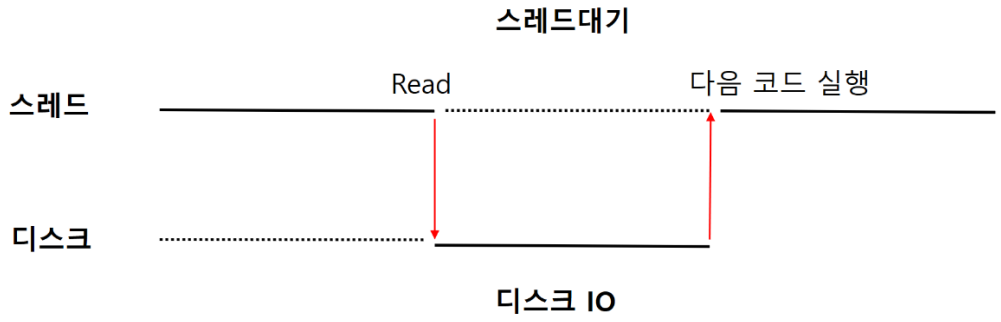
* 작업의 종료 여부를 판단하는 목적으로만 사용하는 것이므로 수동 리셋 이벤트를 사용함.

3. 비동기 호출

1. 동기 방식의 파일 읽기

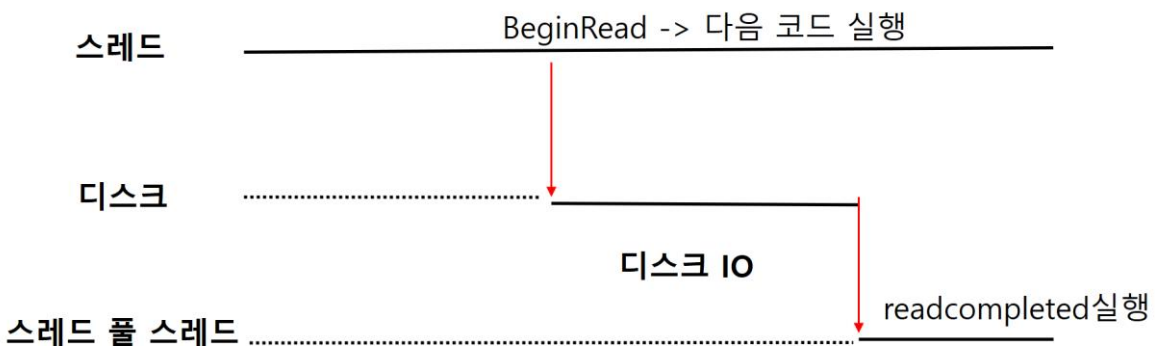
```
using System;
using System.IO;
using System.Text;

class Program
{
    static void Main(string[] args)
    {
        // HOSTS 파일을 읽어서 내용을 출력한다.
        using (FileStream fs = new FileStream(@"C:₩windows₩system32₩drivers₩etc₩HOSTS",
FileMode.Open, FileAccess.Read, FileShare.ReadWrite))
        {
            byte[] buf = new byte[fs.Length];
            fs.Read(buf, 0, buf.Length);
            string txt = Encoding.UTF8.GetString(buf);
            Console.WriteLine(txt);
        }
    }
}
```



2. 비동기 방식의 파일 읽기

```
using System;
using System.IO;
using System.Text;
class FileState{        public byte[] Buffer;          public FileStream File;     }
class Program{
    static void Main(string[] args)      {
        FileStream fs = new FileStream(
                                @"C:₩windows₩system32₩drivers₩etc₩HOSTS",
                                FileMode.Open, FileAccess.Read, FileShare.ReadWrite);
        FileState state = new FileState();
        state.Buffer = new byte[fs.Length];
        state.File = fs;
        fs.BeginRead(state.Buffer, 0, state.Buffer.Length, readCompleted, state);
        // BeginRead 비동기 메서드 호출은 스레드로 곧바로 제어를 반환하기 때문에
        // 이곳에서 자유롭게 다른 연산을 동시에 진행할 수 있다.
        Console.ReadLine();
        fs.Close();
    }
    // 읽기 작업이 완료되면 메서드가 호출된다.
    static void readCompleted(IAsyncResult ar)      {
        FileState state = ar.AsyncState as FileState;
        state.File.EndRead(ar);
        string txt = Encoding.UTF8.GetString(state.Buffer);
        Console.WriteLine(txt);
    }
}
```

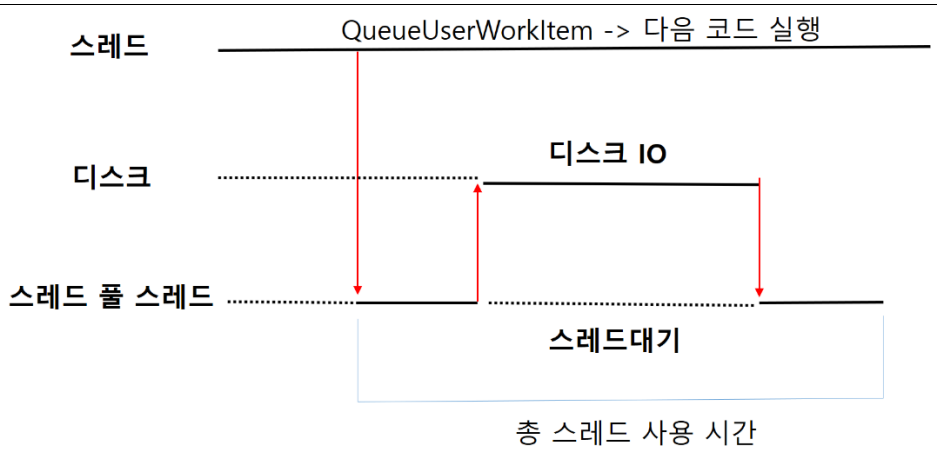


3. 스레드 풀을 직접 사용하는 방식

```
using System;
using System.IO;
using System.Text;
using System.Threading;
class Program{
    static void Main(string[] args)    {
        ThreadPool.QueueUserWorkItem(readCompleted);

        // QueueUserWorkItem 메서드 호출은 곧바로 제어를 반환하기 때문에
        // 이곳에서 자유롭게 다른 연산을 동시에 진행할 수 있다.
        Console.ReadLine();
    }

    // 읽기 작업을 스레드 풀에 대행한다.
    static void readCompleted(object state)    {
        using (FileStream fs = new FileStream(@"C:₩windows₩system32₩drivers₩etc₩HOSTS",
FileMode.Open, FileAccess.Read, FileShare.ReadWrite))
        {
            byte[] buf = new byte[fs.Length];
            fs.Read(buf, 0, buf.Length);
            string txt = Encoding.UTF8.GetString(buf);
            Console.WriteLine(txt);
        }
    }
}
```



총 스레드 사용 시간

* 스레드 풀로부터 빌려온 스레드의 사용 시간이 더 길어졌다.

4. System.Delegate의 비동기 호출

비동기 호출은 입출력 장치와의 속도 차이에서 오는 비효율적인 스레드 사용 문제를 극복하기 위해 사용된다.

닷넷은 입출력 장치 뿐만 아니라 일반 메서드에 대해서도 비동기 호출을 할 수 있는 수단을 제공하는데, 델리게이트가 그러한 역할을 한다.

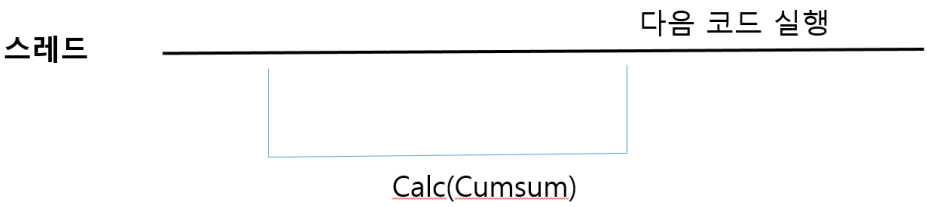
즉, 메서드를 델리게이트로 연결해 두면 이미 비동기 호출을 위한 기반이 마련된 것이나 다름없다.

```
using System;
public class Calc{
	public static long Cumsum(int start, int end)	{
		long sum = 0;
		for (int i = start; i <= end; i++)
		{
			sum += i;
		}
		return sum;
	}
}
class Program{
	public delegate long CalcMethod(int start, int end);

	static void Main(string[] args)	{
		CalcMethod calc = new CalcMethod(Calc.Cumsum);

		long result = calc(1, 100);
		Console.WriteLine(result); // 출력 결과: 5050
	}
}
```

* calc 델리게이트 수행은 현재의 스레드에서 수행된다.

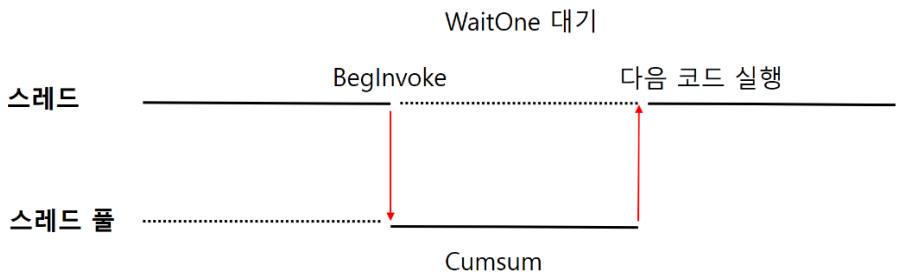


5. 델리게이트의 비동기 호출을 위한 메서드를 사용하면 ThreadPool의 스레드에서 실행

```
using System;
public class Calc{
    public static long Cumsum(int start, int end)    {
        long sum = 0;
        for (int i = start; i <= end; i++)        {
            sum += i;
        }
        return sum;
    }
}
class Program{
    public delegate long CalcMethod(int start, int end);
    static void Main(string[] args)    {
        CalcMethod calc = new CalcMethod(Calc.Cumsum);
        // Delegate 타입의 BeginInvoke 메서드를 호출한다.
        // 이 때문에 Calc.Cumsum 메서드는 ThreadPool의 스레드에서 실행된다.
        IAsyncResult ar = calc.BeginInvoke(1, 100, null, null);

      // BeginInvoke로 반환받은 IAsyncResult 타입의 AsyncWaitHandle 속성은 EventWaitHandle 타입.
      // AsyncWaitHandle 객체는 스레드 풀에서 실행된 Calc.Cumsum의 동작이 완료됐을 때 Signal
상태로 바뀐다.
      // 따라서 아래의 호출은 Calc.Cumsum 메서드 수행이 완료될 때까지 현재 스레드를 대기시킨다.
        ar.AsyncWaitHandle.WaitOne();

        // Calc.Cumsum의 반환값을 얻기 위해 EndInvoke 메서드를 호출한다.
        // 반환값이 없어도 EndInvoke는 반드시 호출하는 것을 권장한다.
        long result = calc.EndInvoke(ar);
        Console.WriteLine(result);
    }
}
```



6. FileStream의 비동기 호출과 유사한 Delegate의 비동기 호출

```
using System;
public class Calc{
	public static long Cumsum(int start, int end)
	{
		long sum = 0;
		for (int i = start; i <= end; i++)
		{
			sum += i;
		}
		return sum;
	}
}

class Program{
	public delegate long CalcMethod(int start, int end);

	static void Main(string[] args)	{
		CalcMethod calc = new CalcMethod(Calc.Cumsum);
		calc.BeginInvoke(1, 100, calcCompleted, calc);
		Console.ReadLine();
	}
	static void calcCompleted(IAsyncResult ar)
	{
		CalcMethod calc = ar.AsyncState as CalcMethod;
		long result = calc.EndInvoke(ar);
		Console.WriteLine(result);
	}
}
```