真的有外星人吗？

假如这个世界上只剩下你一个人，当你正坐在屋子里的时候，这时突然响起了敲门声，那么会是谁呢？

C#线程系列讲座(3)：线程池和文件下载服务器

本文为原创，如需转载，请注明作者和出处，谢谢！

上一篇：C#线程系列讲座(2)：Thread类的应用

如果设计一个服务器程序，每当处理用户请求时，都开始一个线程，将会在一定程序上消耗服务器的资源。为此，一个最好的解决方法就是在服务器启动之前，事先创建一些线程对象，然后，当处理客户端请求时，就从这些建好的线程中获得线程对象，并处理请求。保存这些线程对象的结构就叫做线程池。

在C#中可以通过System.Threading.ThreadPool类来实现，在默认情况下，ThreadPool最大可建立500个工作线程和1000个I/O线程（根据机器CPU个数和.net framework版本的不同，这些数据可能会有变化）。下面是一个用C#从线程池获得线程的例子:

```
private static void execute(object state)
{
    Console.WriteLine(state);
}
static void Main(string[] args)
{

    int workerThreads;
    int completionPortThreads;

    ThreadPool.GetMaxThreads(out workerThreads, out completionPortThreads);
    Console.WriteLine(workerThreads);
    Console.WriteLine(completionPortThreads);
    ThreadPool.QueueUserWorkItem(execute,"线程1");    // 从线程池中得到一个线程，并运行execute
    ThreadPool.QueueUserWorkItem(execute, "线程2");
    ThreadPool.QueueUserWorkItem(execute, "线程3");
    Console.ReadLine();
}
```

下图为上面代码的运行结果。

< 2008年7月 >

日	一	二	三	四	五	六
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

导航

博客园
首页
新随笔
联系
订阅XML
管理

统计

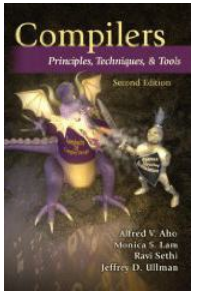
随笔 - 85
文章 - 0
评论 - 650
引用 - 29

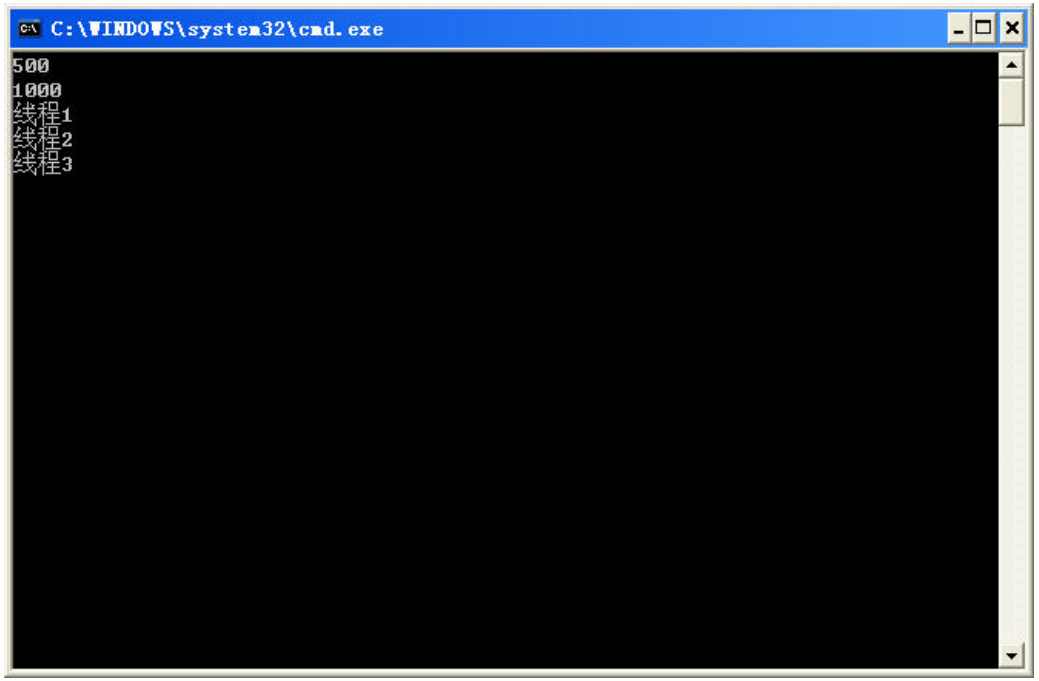
公告

我的其他Blog

http://nokiaguy.blogjava.net
http://blog.csdn.net/nokiaguy

正在读的书





要注意的是，使用ThreadPool获得的线程都是后台线程。

下面的程序是我设计的一个下载文件服务器的例子。这个例子从ThreadPool获得线程，并处理相应的客户端请求。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Net.Sockets;
using System.IO;

namespace MyThread
{
    class FileServer
    {
        private String root;
        private Thread listenerThread;

        private void worker(object state)
        {
             TcpClient client = state as TcpClient;
             try
             {

                 client.ReceiveTimeout = 2000;
                 Stream stream = client.GetStream();
                 System.IO.StreamReader sr = new StreamReader(stream);
```

```
                String line = sr.ReadLine();
                String[] array = line.Split(' ');
                String path = array[1].Replace('/', '\\');
                String filename = root + path;
                if (File.Exists(filename))  // 如果下载文件存在，开始下载这个文件
                {
                    FileStream fileStream = new FileStream(filename, FileMode.Open, FileAccess.Read,
                                                           FileShare.Read);
                    byte[] buffer = new byte[8192]; // 每次下载8K
                    int count = 0;
                    String responseHeader = "HTTP/1.1 200 OK\r\n" +
                                            "Content-Type:application/octet-stream\r\n" +
                                            "Content-Disposition:attachment;filename=" +
                                                  filename.Substring(filename.LastIndexOf("\\") + 1
) + "\r\n\r\n";
                    byte[] header = ASCIIEncoding.ASCII.GetBytes(responseHeader);
                    stream.Write(header, 0, header.Length);
                    while ((count = fileStream.Read(buffer, 0, buffer.Count())) > 0)
                    {
                        stream.Write(buffer, 0, count);
                    }
                    Console.WriteLine(filename + "下载完成");
                }
                else  // 文件不存在，输出提示信息
                {
                    String response = "HTTP/1.1 200 OK\r\nContent-Type:text/plain;charset=utf-8\r\n\
r\n文件不存在";
                    byte[] buffer = ASCIIEncoding.UTF8.GetBytes(response);
                    stream.Write(buffer, 0, buffer.Length);
                }

            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            finally
            {
                if (client != null)
                {
                    client.Close();
                }
            }
        }

        private void listener()
        {
            TcpListener listener = new TcpListener(1234);
            listener.Start();  // 开始监听客户端请求
            TcpClient client = null;
```

```
            while (true)
            {
                client = listener.AcceptTcpClient();
                client.ReceiveTimeout =2000;
                ThreadPool.QueueUserWorkItem(worker, client);  // 从线程池中获得一个线程来处理客户端请求
            }
        }
        public FileServer(String root)
        {
            this.root= root;
        }
        public void start()
        {
            listenerThread = new Thread(listener);
            listenerThread.Start();  // 开始运行监听线程
        }
    }
}
```

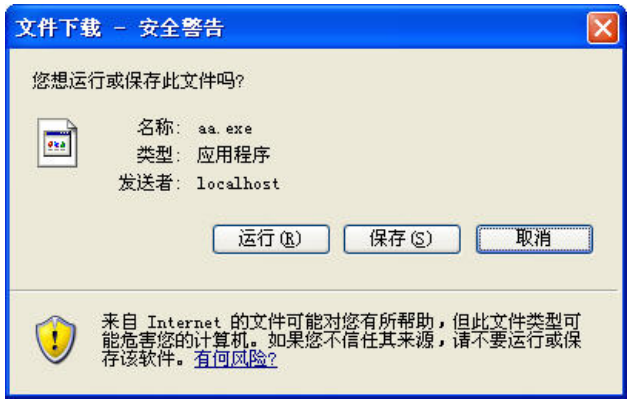
FileServer类的使用方法：

FileServer fs = new FileServer("d:\\download");

fs.start(); // 端口为1234

如果d:"download目录中有一个叫aa.exe的文件，在浏览器中输入如下的地址可下载：
http://localhost:1234/aa.exe

下图为下载对话框：



要注意的是，本程序并没有处理含有中文和其他特殊字符（如空格）的url，因为，文件名要为英文名（不能有空格等特殊字符）。

下一篇：C#线程系列讲座(4)：同步与死锁

2008年5月 (36)
2008年4月 (1)

相册
images
其他
微软认证

Blogs
anytao
leo zhang（职业规划师）
刘江（图灵主编）

开源
微软开源网站

协议
MSN Messenger

积分与排名
积分 - 145137
排名 - 381

最新评论 XML

1. Re:.net framework3.5新特性2：var、初始化、匿名类和扩展方法
我看了文章和大家的说评语，楼主写的也不错，大家说也很好，让我对var又加深对var的了解。
--Dean123

2. Re:全排列算法原理和实现
回8楼和9楼的，在permut函数for循环中，调用完permut之后还要执行一次swap的，要把数组还原成未调用permut之前的状态，保证下一次循环或者上一级循环swap数组元素的正确。愚见愚见。...
--Little_Angel

3. Re:移动的MobileMarket个人终于可以上传软件了
还是即将上线，敬请期待~？~？
--DAP

4. Re:移动的MobileMarket个人终于可以上传软件了
哦，好久没去关注这方面了，去看看
--peterzb

5. Re:想抢先体验Android操作系统的魅力吗？那就使用Android LiveCD吧！
买不起，只有这样体验下了~~~
--J.Motto

阅读排行榜
1. 全排列算法原理和实现(6165)
2. C#线程系列讲座(1)：BeginInvoke和En

《银河系列原创教程》发布

posted on 2008-07-18 19:22 银河使者 阅读(2995) 评论(24) 编辑 收藏 网摘 所属分类: 进程、线程、并发, C#, .net高级技术, 原创

评论

#1楼 2008-07-18 20:09 Angel Lucifer

```
/*
* 在默认情况下，ThreadPool最大可建立500个工作线程和1000个I/O线程
*/
```

在 .NET 2.0 中， ThreadPool 中默认最大工作线程是每 CPU 25个。
在 .NET 3.5 中， ThreadPool 中默认最大工作线程是每 CPU 250个。

这个变化是为了处理线程池中的死锁，但无法彻底排除死锁的可能性，只是让该问题的发生概率大大减小而已。

由此，可以推知楼主的机器是双核 CPU，安装了 .NET Framework 3.5 或者 .NET Framework 2.0 SP1，:-)。 回复 引用 查看

#2楼[楼主] 2008-07-18 20:12 银河使者
没错，忘说了，是双核，应该说一下。哈哈。 回复 引用 查看

#3楼[楼主] 2008-07-18 20:18 银河使者
我想增大工作线程数主要是为了在默认情况下可同时处理更多的工作。至于死锁，是要用同步解决的，和线程池中的线程个数无关。再说ThreadPool最然是500个工作线程，但也是一开始一个一个起的（虽然可以设置），不是一起都运行时。就算是50000个线程也同样会发生dead lock。 因为死锁是相关的一些线程之间协作不好而造成的，和其他无关的线程没有任何关系。 回复 引用 查看

#4楼 2008-07-18 20:33 Angel Lucifer

仅部分同意，这个问题有相关的说明，线程数并非越多越好，呵呵。

当线程池有太多线程等待其他任务结束时就会出现死锁：一旦所有25个线程都被阻塞的时候，等待中的任务就无法分配到线程了。250只是降低了这种死锁的可能。

回复 引用 查看

#5楼[楼主] 2008-07-18 20:52 银河使者
所谓死锁是两个或多个线程互相等待造成的，并不是线程少了就容易死锁，多了就不死了。 还有死锁也是不无法分配线程造成的，那也不叫死锁，所果在线程池中分配不到线程，这个任务就会被放到等待队列中，直接有空闲thread来处理它，或者在超时时间后被丢弃。

死锁其实就是两个或多个线程互相被锁住了。如下面代码（只是伪代码，大概意识）如示：

```
thread1
{
lock(a)
{
// 假设thread1刚好执行到这被thread2把cpu夺走了
lock(b) { ... }
}
```

```
}

thread2
{
lock(b)
{
// 假设thread2刚好执行到这被thread1把cpu夺走了
lock(a) { ... }
}
}
```

我们可以想象，如果thread1和thread2正好执行到了上面代码注释的部分。那么thread1中要执行lock(b)，而thread2中要执行lock(a)，但这时a和b都被锁住了，所有就死了。 当然，解决这个问题的方法也非常简单，就是thread1和thread2的lock语句顺序保持一致，代码如下：

```
thread1
{
lock(a)
{

lock(b) { ... }
}
}

thread2
{
lock(a)
{
lock(b) { ... }
}
}
```

上面代码是永远不会发生死锁地。
回复 引用 查看

#6楼 2008-07-18 21:12 airwolf2026
呵呵.还行. 回复 引用 查看

#7楼 2008-07-18 21:32 lexus
如果有100000个人同时下载文件，那会出现什么情况？从第501个开始服务器没有响应，是这样吗？ 回复 引用 查看

#8楼[楼主] 2008-07-18 21:38 银河使者
如果下载到501个文件时，页前500个都没下载完，服务器会处理等待状态，在客户端会看到浏览器的进度条正在前进，但不会出现下载对话框，直到有其他的自由线程。或者由于超时而被server抛弃这个任务。 回复 引用 查看

#9楼 2008-07-18 22:26 Angel Lucifer
@银河使者
我当然清楚死锁的概念。.NET 3.5 之所以从 25 提高到 250的目的是为了降低发生死锁的概率，而不是说它一定会发生死锁，呵呵。 回复 引用 查看

#10楼 2008-07-18 22:28 Angel Lucifer
避免死锁主要还是开发人员的责任，因为线程池并不知道开发人员如何同步。

此外，如果使用 Lock-Free ，则根本不会有死锁的可能。 回复 引用 查看

#11楼 2008-07-19 01:39 jv9
good topic 回复 引用 查看

#12楼[楼主] 2008-07-19 07:49 银河使者
只有可能发生死锁的任务在不同的线程中同时运行，就可能会发生死锁。 如果这些线程不同时运行，当然就会不发生死锁。
但不知将25增到250可以降低死锁发生的概率是从何说起，怎么将可同时运行的线程数增大就会降低死锁的概率呢？请@ Angel Lucifer 明示 回复 引用 查看

#13楼 2008-07-19 08:56 雅阁布
up!!! 回复 引用 查看

#14楼 2008-07-19 10:09 Angel Lucifer
@银河使者
文字描述太麻烦，用一段代码来说明吧。

回复 引用 查看

#15楼 2008-07-19 10:32 Angel Lucifer

```
using System.Threading;
namespace Lucifer.CSharp.Sample
{
    class Program
    {
        public static void Main()
        {
            ThreadPool.SetMaxThreads(1, 1);

            WaitCallback callback = delegate { MethodA(); };
            callback.BeginInvoke(null, null, null);

            Console.ReadLine();
        }

        private static void MethodA()
        {
            Console.WriteLine("Thread A is beginning…");

            WaitCallback callback = delegate { MethodB(); };
            IAsyncResult result = callback.BeginInvoke(null, null, null);

            //模拟实际工作。
            Console.WriteLine("1…");
            Console.WriteLine("2…");
            Console.WriteLine("3…");

            //这里永远也不会返回。
            callback.EndInvoke(result);
        }

        private static void MethodB()
        {
            //永远也不会运行到这里，因为线程池线程耗尽，导致资源竞争引发了死锁。
            Console.WriteLine("Thread B is beginning…");
```

```
        }
    }
}
```

回复 引用 查看

#16楼 2008-07-19 10:35 Angel Lucifer
本来写了一些解释，结果 Firefox 下老出问题，搞的俺灰心了。

琢磨一下吧，:-)
回复 引用 查看

#17楼[楼主] 2008-07-19 11:53 银河使者
@Angel Lucifer
不好意思，你的这段代码会死锁呢？我在机器上调试，并没有死锁啊，

这种用方式是由于没有线程执行任务导致的死锁，但不知怎么着，在我的机器上没死，会不会是因为双核的缘故。 回复 引用 查看

#18楼[楼主] 2008-07-19 11:55 银河使者
再加一层也没有死：

```
public static void Main()
{
ThreadPool.SetMaxThreads(1, 1);

WaitCallback callback = delegate { MethodA(); };
callback.BeginInvoke(null, null, null);

Console.ReadLine();
}

private static void MethodA()
{
Console.WriteLine("Thread A is beginning");

WaitCallback callback = delegate { MethodB(); };
IAsyncResult result = callback.BeginInvoke(null, null, null);

//模拟实际工作。
Console.WriteLine("1");
Console.WriteLine("2");
Console.WriteLine("3");

//这里永远也不会返回。
callback.EndInvoke(result);
}

private static void MethodB()
{
//永远也不会运行到这里，因为线程池线程耗尽，导致资源竞争引发了死锁。
Console.WriteLine("Thread A is beginning");

WaitCallback callback = delegate { MethodC(); };
IAsyncResult result = callback.BeginInvoke(null, null, null);
```

```
//模拟实际工作。
Console.WriteLine("1");
Console.WriteLine("2");
Console.WriteLine("3");

//这里永远也不会返回。
callback.EndInvoke(result);
}
private static void MethodC()
{
Console.WriteLine("methodc");
}
```

回复 引用 查看

#19楼[楼主] 2008-07-19 11:57 银河使者
能解释这是为什么吗？感觉ThreadPool.SetMaxThreads(1, 1);没起作用。

但这种死种方式只是原因之一，而更复杂的死锁是在运行线程之间的死锁。
回复 引用 查看

#20楼 2008-07-19 13:16 Angel Lucifer
忘记你是双核的了，呵呵。

ThreadPool.SetMaxThreads 不能设置最大线程数小于计算机的处理器数目。因为设置不成功，所以还是默认的每CPU 250。

双核情况下，可以考虑设置成 2 或者以上，只要它能返回 true 。

代码可以考虑使用你自己的嵌套三层或者 N 层。

PS：在实际的异步编程中，可能会大量使用 ThreadPool.QueueUserWorkItem 或者异步委托(它基于线程池实现)。如果不注意，很快就会超过 25 导致死锁。我们很难避免这种情况不发生。这正是线程池中线程数量从 25 提升到 250 的原因。

再次PS: 锁机制相当麻烦，一不小心，就会翻船。.NET ThreadPool 的实现也不是十分高效。 回复 引用 查看

#21楼[楼主] 2008-07-19 13:29 银河使者
没错，需要设成 ThreadPool.SetMaxThreads(2, 2);methodc才无法调用。
但你的这种产生死锁的方式是通过在线程里启动另外一个线程而造成的，而且在线程中运行另外一个线程，还必须要调用join或EndXXX，必须要满足这两个条件死锁才会发生。 而且这种死锁也非常好查找，但只要不这样做，也就是不在一个线程里运行另一个线程，就算是运行另外一个线程，也不去调用EndXXX或join，这样则可完全避免这种方式的死锁了，最闹心的就是因同步而产生的死锁。这种死锁方式是防不胜防啊。

本文提供的文件下载服务器是绝对不会死锁发生的，顶多由于线程过多，把下载任务给扔了。 回复 引用 查看

#22楼 2008-07-19 16:41 毁于随
很好的文章.周一再细细看看. 回复 引用 查看

#23楼 2008-07-19 22:31 农[未注册用户]
--引用----------

本文提供的文件下载服务器是绝对不会死锁发生的，顶多由于线程过多，把下载任务给扔了。

不知道您老为什么突然这么说，对于单线程的东西会死锁？呵呵。

死锁和线程池的大小关系不大，而是Handler是否被另一只关联等待的线程互相等待。前面有人已经说了。。。

回复 引用

#24楼[楼主] 2008-07-19 22:52 银河使者

死锁和线程池的大小关系不大，而是Handler是否被另一只关联等待的线程互相等待。前面有人已经说了。。。

这是我说的。两个线程互相等待。还举了例子，而另外一个人举了另外一个死锁的例子，就是一个线程里调用另一个线程时的死锁。

我的文件服务器是多线程，不是单线程地， 只是线程之间没有关联，如是不会死锁，这好象没错吧。

至于死锁和线程池的关系，有，但不大。它们之间的关系我也是听@Angel Lucifer说的。 主要是线程池会限制线程运行的个数，所以会出现Angel Lucifer描述的这种情况。 回复 引用 查看

刷新评论列表 刷新页面 返回页首

发表评论

昵称: [登录] [注册]

主页:

邮箱: (仅博主可见)

评论内容: 闪存 个人主页

提交 登录 注册

[使用Ctrl+Enter键快速提交评论]

个人主页上线测试中
今天你闪了吗？