实 验 十一　互斥量与管道通信

11.1　实验目的

进程间有多种通讯方式，互斥量实现对资源的唯一访问，管道提供大数据量的传输，本次实验使用互斥量实现对共享变量的访问，并通过有名管道机制进行进程间通信。

11.2　互斥量

计算机中的互斥量称为 Mutex，在同一时间最多只能有一个线程可以获取并拥有它，它适合不同线程对同一共享资源互斥访问的应用场合，例如对全局变量，同一个文件或者是数据库同一个对象的访问等，设置程序先获得互斥量再访问共享资源。互斥量是操作系统管理的内核对象，能够在不同进程中发挥作用，这些进程通过互斥量的名称来访问它。互斥量可保证数据操作的完整性。

11.2.1　使用互斥量

互斥量在使用时需要考虑线程对它的所属权，当线程在创建互斥量对象时可指定是否具有对其拥有，最常使用的 WaitOne 方法，如果等待成功时也获得所有性，当线程使用 ReleaseMutext 方法主动放弃对其所有时，其它线程才有机会获取它，否则其它等待互斥量对象的线程将保持阻止。下面是使用互斥量的代码：

```
bool createNew;
m = new Mutex(false, "MutexExample", out createNew);
if (createNew)
{
  //互斥量不存在，创建新实例
  textBox1.AppendText(" 创建互斥量 MutexExample\r\n");
  textBox1.ScrollToCaret();
}
else {
  //互斥量已存在
  textBox1.AppendText(" 互斥量 MutexExample 已存在 \r\n");
  textBox1.ScrollToCaret();
}
  //互斥量存在，获取拥有
```

```
    m.WaitOne();
    textBox1.AppendText(" 拥有互斥量 MutexExample\r\n");
    textBox1.ScrollToCaret();
```

一个具体的互斥量对象只能被一个线程拥有，但是线程可调用多次的 WaitOne 方法重复对其所有，使用 ReleaseMutex 方法释放对互斥量所属权，而每一个成功的 WaitOne 方法对应一次 ReleaseMutex，假如互斥量在不拥有互斥量对象的情况下调用 ReleaseMutex 方法将引发 ApplicationException 异常。如果线程调用的 ReleaseMutex 方法与 WaitOne 方法数不匹配，而线程运行终止 mutex 被放弃，这种程序逻辑是错误的；在这种情况下的互斥量仍可被其它线程取得所属权，但会获得 AbandonedMutexException 异常，这表明了互斥量被上一个线程未释放互斥体时便退出。本实验中对互斥量的操作是通过窗体线程的，下面这段代码窗体线程释放互斥量的所属权：

```
bool createNew;
m = new Mutex(false, "MutexExample", out createNew);
try
{
    m.ReleaseMutex();
    textBox1.AppendText(" 释放互斥量 MutexExample\r\n");
    textBox1.ScrollToCaret();
}
catch (ApplicationException)
{
    textBox1.AppendText(" 当前线程不拥有互斥量 MutexExample，无法执行释放操作 \r\n");
    textBox1.ScrollToCaret();
}
```

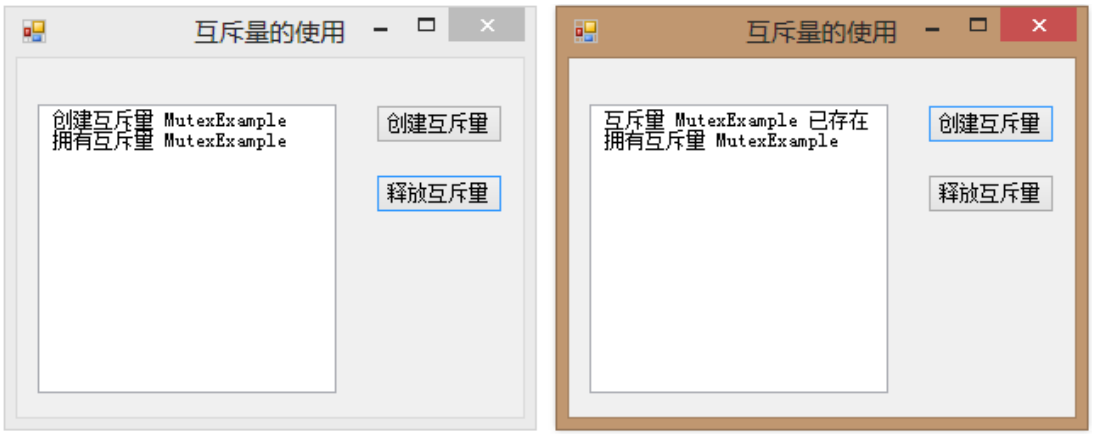

图 11-1 互斥量的使用

代码编译后生成程序，启动它的两个实例，互斥量对象被某个窗体线程获取后则另一个窗体处于等待状态，这将造成这个窗体没有响应。

11.2.2 用互斥量控制程序只运行一次

互斥量 Mutex 有个最常见的用途：用于控制一个应用程序只能有一个实例运行。原理是当程序启动时创建一个互斥量对象，当程序再次启动时通过检查互斥量对象是否已经被创建得知是否已有程序实例运行。在程序的 Program.cs 文件中修改后的代码如下：

```
bool firstInstance;
mutex = new Mutex(true, @"Global\MutexSampleApp", out firstInstance);
if (firstInstance)
{
  Application.EnableVisualStyles();
  Application.SetCompatibleTextRenderingDefault(false);
  Application.Run(new Form1());
}
else
{
  MessageBox.Show(" 已经有本程序的实例运行，不再创建新实例");
}
```

11.3 管道通信

分别建立两个窗体应用程序，设位置为 D:\xue，名称分别为 pipe_s 和 pipe_c 作为管道的服务端和客户端程序。

11.3.1 管道通信说明

管道机制是计算机提供的一种进程间通信 (IPC) 方式，由操作系统创建管道对象，发送进程可以向管道写入数据，接收进程由管道中读出数据。管道还可进行跨计算机的通信，可使用网络，也可使用文件等，它屏蔽低层实现机制提供给进程通信机制。有两种形式管道，有名管道与无名管道，无名管道只在单一进程内使用；有名管道则以" 文件名" 来标识，可用于不同进程间的通信；进程的重定向就是管道的一种应用形式，命名管道用于网络编程方案使用时，它实际上建立一个简单的客户机 / 服务器数据通信体系，可在其中可靠地传输数据。命名管道具有很好的使用灵活性，它的表现有：

1. 既可用于本地，又可用于网络。
2. 通过它的名称而被引用。
3. 支持多客户机连接。
4. 支持双向通信。
5. 支持异步重叠 I/O 操作。

本实验利用有名管道进行两个进程间的通信。

11.3.2 使用命名管道

命名管道的使用与 Windows 文件系统有很大关系，它使用" 命名管道文件系统"(Named Pipe File System，NPFS)，用户程序可像使用文件一样对命名管道进行数据读写，而无须关心低层传送协议。命名管道用" 通用命名规范"(UNC) 来标识的，如下面：

\\server\pipe\[path]name

在这个命名中，\\server 指定服务器名字；\pipe 是固定部分，表示这是管道对象；\[path]name 则唯一标识管道对象的名字。

在.NET 平台中提供对命名管道的支持，实现命名或者匿名管道的类型包含在 System.IO.Pipes 命名空间中，类如表11-1

表 11-1 管道类

类名	功能说明
AnonymousPipeClientStream	匿名管道流的客户端，该管道流既支持同步和异步读写操作。
AnonymousPipeServerStream	匿名管道流的服务端，该管道流既支持同步和异步读写操作。
NamedPipeClientStream	命名管道流的客户端，该管道流既支持同步和异步读写操作。
NamedPipeServerStream	命名管道流的服务端，该管道流既支持同步和异步读写操作。
PipeAccessRule	使用访问控制项 (ACE) 定义管道的访问规则。
PipeAuditRule	使用访问控制项 (ACE) 定义管道的审核规则。
PipeSecurity	管道的访问控制和审核安全。
PipeStream	同时支持匿名管道和命名管道。

命名管道提供了两种基本通信模式：字节模式和消息模式。在字节模式中，数据以连续的字节流的形式，在客户机与服务器之间流动。在消息模式中，客户机和服务器则通过一系列不连续的数据单位，进行数据的收发。在创建有名管道时 PipeTransmissionMode.Byte 枚举值表示管道内数据以字节流进行传输，PipeTransmissionMode.Message 指示管道内数据作为消息流进行传输。

11.3.3 命名管道通信流程

命名管道 Server 和 Client 间通信的实现流程，(1) 建立连接：服务端创建一个命名管道服务端实例，NamedPipeServerStream 类的 WaitForConnection 方法用以侦听来自客户端的连接请求，该函数将阻塞线程。当客户端调用 Connect 方法连接后，双方即可进行通信。(2) 通信实现：建立连接之后，客户端与服务器端即可通过 Stream 流的 Read 方法和 Write 方法进行读写通信。(3) 连接终止：当客户端与服务端的通信结束，或由于某种原因一方需要断开时，可使用 Close 方法关闭管道连接。

11.3.4 命名管道服务端实现

下面的代码实现管道服务端代码：

NamedPipeServerStream pipeServer = new NamedPipeServerStream("testpipe", PipeDirec-

```
tion.Out);
  textBox1.Text = " 等待客户连接";
  pipeServer.WaitForConnection();
  try
  {
    StreamWriter sw = new StreamWriter(pipeServer);
    sw.AutoFlush = true;
    sw.WriteLine(" 通过管道来到的信息");
    sw.Close();
  }
  catch (IOException e1)
  {
    MessageBox.Show("ERROR: {0}", e1.Message);
  }
```

11.3.5 命名管道客户端实现

下面的代码实现管道客户端代码：

```
NamedPipeClientStream pipeClient =new NamedPipeClientStream(".",
            "testpipe", PipeDirection.In);
pipeClient.Connect();
StreamReader sr = new StreamReader(pipeClient);
textBox1.Text=sr.ReadToEnd();
```

以上代码实现管道的同步读写操作，可使用异步方式获得更好的程序响应方式。

11.4 作业

1. 查阅 MSDN 获取 NamedPipeClientStream 类的使用方法，实现管道的异步读写。
2. 使用互斥量实现程序只能运行一个实例。