```csharp
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.Threading;

namespace ServerAsyn
{
    //
    /// <summary>
    ///State object for reading client data asynchronously
    /// </summary>
    public class StateObject
    {
        // Client  socket.
        public Socket workSocket = null;
        // Size of receive buffer.
        public const int BufferSize = 1024;
        // Receive buffer.
        public byte[] buffer = new byte[BufferSize];
        // Received data string.
        public StringBuilder sb = new StringBuilder();
    }

    public class AsynchronousSocketListener
    {
        // Thread signal.ManualResetEvent 允许线程通过发信号互相通信。通常，此通信涉及一个线程在其他线程进行之前必须完成的任务。
        //当一个线程开始一个活动（此活动必须完成后，其他线程才能开始）时，它调用 Reset 以将 ManualResetEvent 置于非终止状态。
        //此线程可被视为控制 ManualResetEvent。 调用 ManualResetEvent 上的 WaitOne 的线程将阻止，并等待信号。
        //当控制线程完成活动时，它调用 Set 以发出等待线程可以继续进行的信号。 并释放所有等待线程。
        public static ManualResetEvent allDone = new ManualResetEvent(false);

        public AsynchronousSocketListener()
        {
        }

        public static void StartListening()
        {
            // Data buffer for incoming data.
            byte[] bytes = new Byte[1024];

            IPAddress ipAddress = IPAddress.Parse("127.0.0.1"); ;
            IPEndPoint localEndPoint = new IPEndPoint(ipAddress, 9500);

            // Create a TCP/IP socket.
            Socket listener = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);

            // Bind the socket to the local endpoint and listen for incoming connections.
            try
            {
                listener.Bind(localEndPoint);
                listener.Listen(100);//允许队列等待100个连接

                while (true)
                {
                    // Set the event to nonsignaled state.
                    allDone.Reset();//初始化信号量，今后所有调用allDone.waitone的线程将阻止

                    // Start an asynchronous socket to listen for connections.
                    Console.WriteLine("Waiting for a connection...");

                    //使用 AsyncCallback 委托在一个单独的线程中处理异步操作的结果，并将listener作为参数传递给方法
                    listener.BeginAccept(new AsyncCallback(AcceptCallback),listener);

                    // Wait until a connection is made before continuing.
                    allDone.WaitOne();//当前线程阻塞
                }

            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }

            Console.WriteLine("\nPress ENTER to continue...");
            Console.Read();

        }

        public static void AcceptCallback(IAsyncResult ar)
        {
            //当接收到连接请求时，本方法被自动调用
            // Signal the main thread to continue.
            allDone.Set();//所有已经阻止的allDone将继续

            // Get the socket that handles the client request.
            Socket listener = (Socket)ar.AsyncState;
            Socket handler = listener.EndAccept(ar);

            // Create the state object.用来处理后续数据
            StateObject state = new StateObject();
            state.workSocket = handler;

            //注册接收数据的方法
            handler.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,new AsyncCallback(ReadCallback), state);
            Console.WriteLine("建立连接，等待接收数据。。");
        }

        public static void ReadCallback(IAsyncResult ar)
        {
            String content = String.Empty;

            // Retrieve the state object and the handler socket
            // from the asynchronous state object.
            StateObject state = (StateObject)ar.AsyncState;
            Socket handler = state.workSocket;

            Console.WriteLine("开始接收数据...");

            // Read data from the client socket.
            int bytesRead = handler.EndReceive(ar);

            if (bytesRead > 0)
            {
                // There  might be more data, so store the data received so far.
                state.sb.Append(Encoding.ASCII.GetString(state.buffer, 0, bytesRead));

                // 检查是否输入结束
                content = state.sb.ToString();
                if (content.IndexOf("\r") > -1)
                {
                    // All the data has been read from the
                    // client. Display it on the console.
                    Console.WriteLine("Read {0} bytes from socket. \n Data : {1}",
                        content.Length, content);
                    // Echo the data back to the client.
                    Send(handler, content);
                }
                else
                {
                    // Not all data received. Get more.
                    handler.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,
                    new AsyncCallback(ReadCallback), state);
                }
            }
        }

        private static void Send(Socket handler, String data)
        {
            // Convert the string data to byte data using ASCII encoding.
            byte[] byteData = Encoding.ASCII.GetBytes(data);

            // Begin sending the data to the remote device.
            handler.BeginSend(byteData, 0, byteData.Length, 0,
                new AsyncCallback(SendCallback), handler);
        }

        private static void SendCallback(IAsyncResult ar)
        {
            try
            {
                // Retrieve the socket from the state object.
                Socket handler = (Socket)ar.AsyncState;

                // Complete sending the data to the remote device.
                int bytesSent = handler.EndSend(ar);
                Console.WriteLine("Sent {0} bytes to client.", bytesSent);

                handler.Shutdown(SocketShutdown.Both);
                handler.Close();

            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
        }


        public static int Main(String[] args)
        {
            StartListening();
            return 0;
        }
    }
}

```