.Net Sockets

Jim Fawcett

CSE681 – Software Modeling & Analysis

Fall 2008

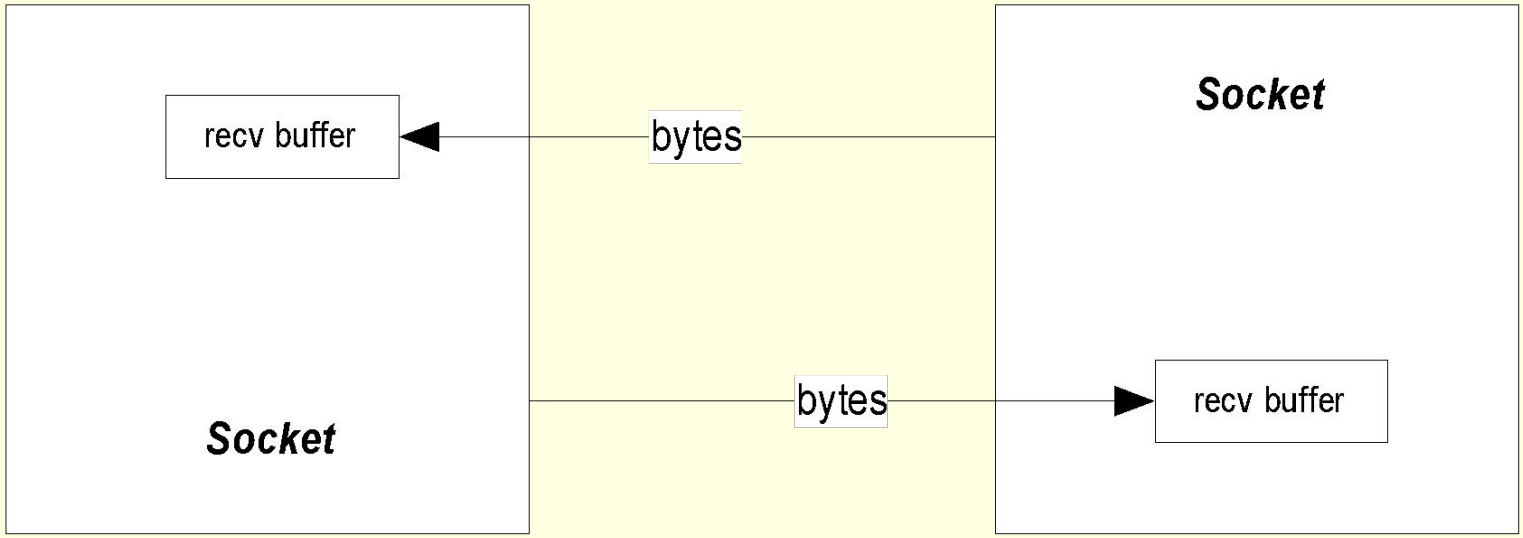
References

- www.msdn.microsoft.com/library
  - .Net Development/.Net Framework SDK/
    - .Net Framework/Reference/ClassLibrary/
      - System.Net.Sockets
- http://www.dotnetjunkies.com/quickstart/howto/doc/TCPUDP/DateTimeClient.aspx
- C# NetworkProgramming, Richard Blum, Sybex, 2003
- Win32 Sockets, Jim Fawcett, Fall 2002

What are Sockets?

- Sockets provide a common interface to the various protocols supported by networks.
- They allow you to establish connections between machines to send and receive data.
- Sockets support the simultaneous connection of multiple clients to a single server machine.

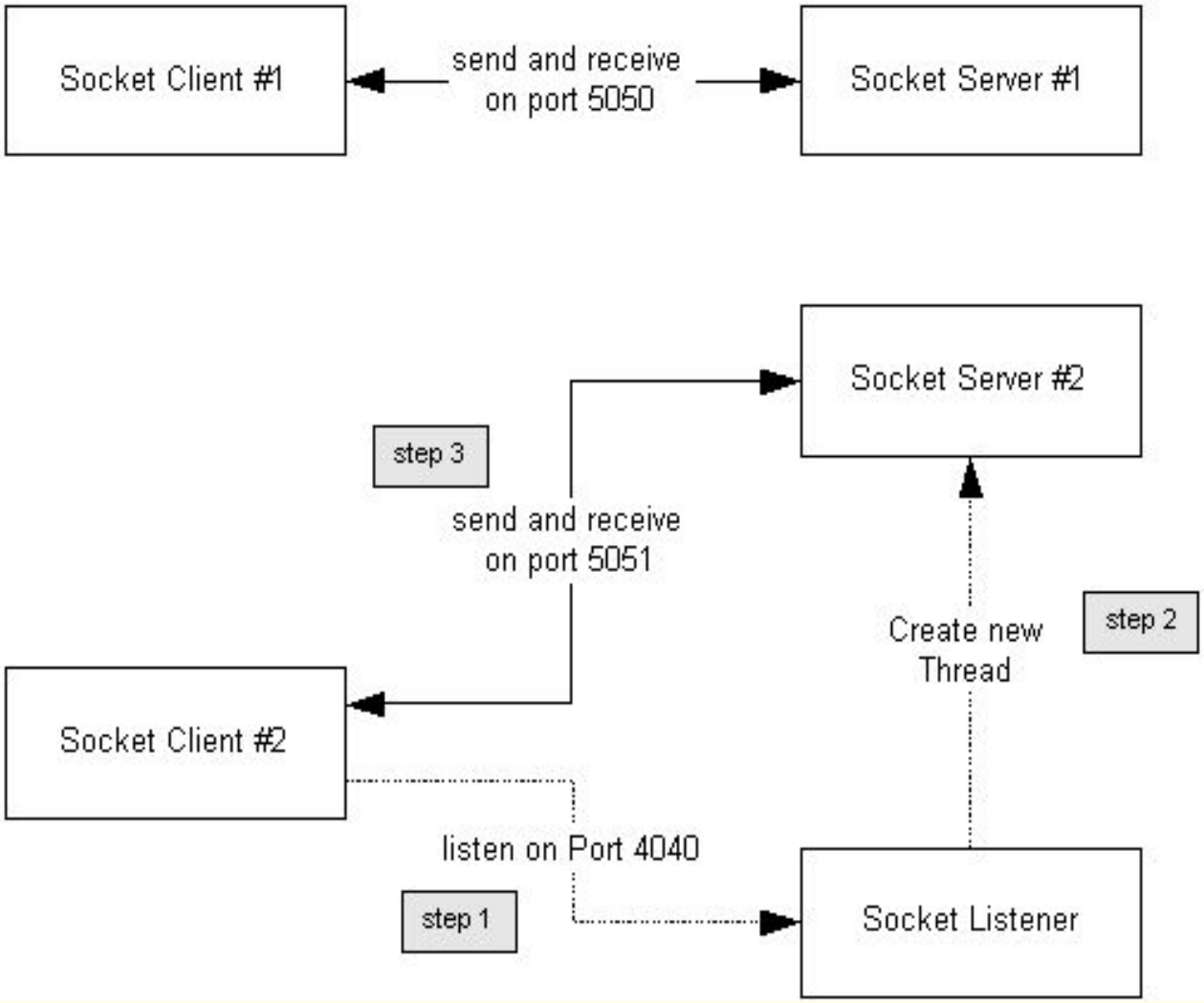
Socket Logical Structure



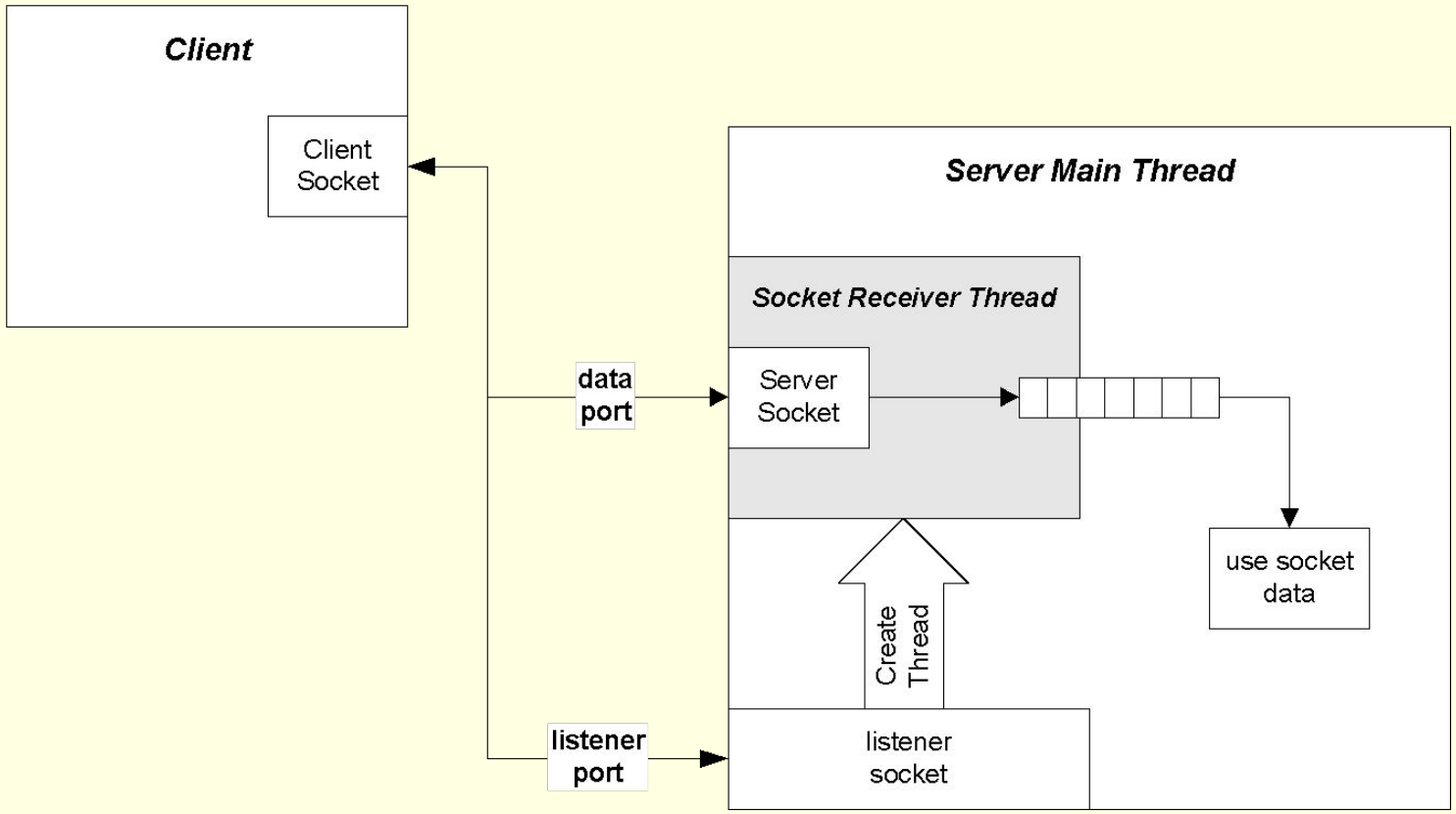
How do Sockets Function?

- There are several modes of operation available for sockets.
  - A very common mode is to establish a socket listener that listens on some port, say 4040, for connection requests.
  - When a socket client, from another process or a remote computer, requests a connection on port 4040, the listener spawns a new thread that starts up a socket server on a new port, say 5051.
  - From that time on the socket client and socket server communicate on port 5051. Either one can send data, in the form of a group of bytes, to the other.
  - Meanwhile the listener goes back to listening for connection requests on port 4040.

Socket Client, Server, and Listener



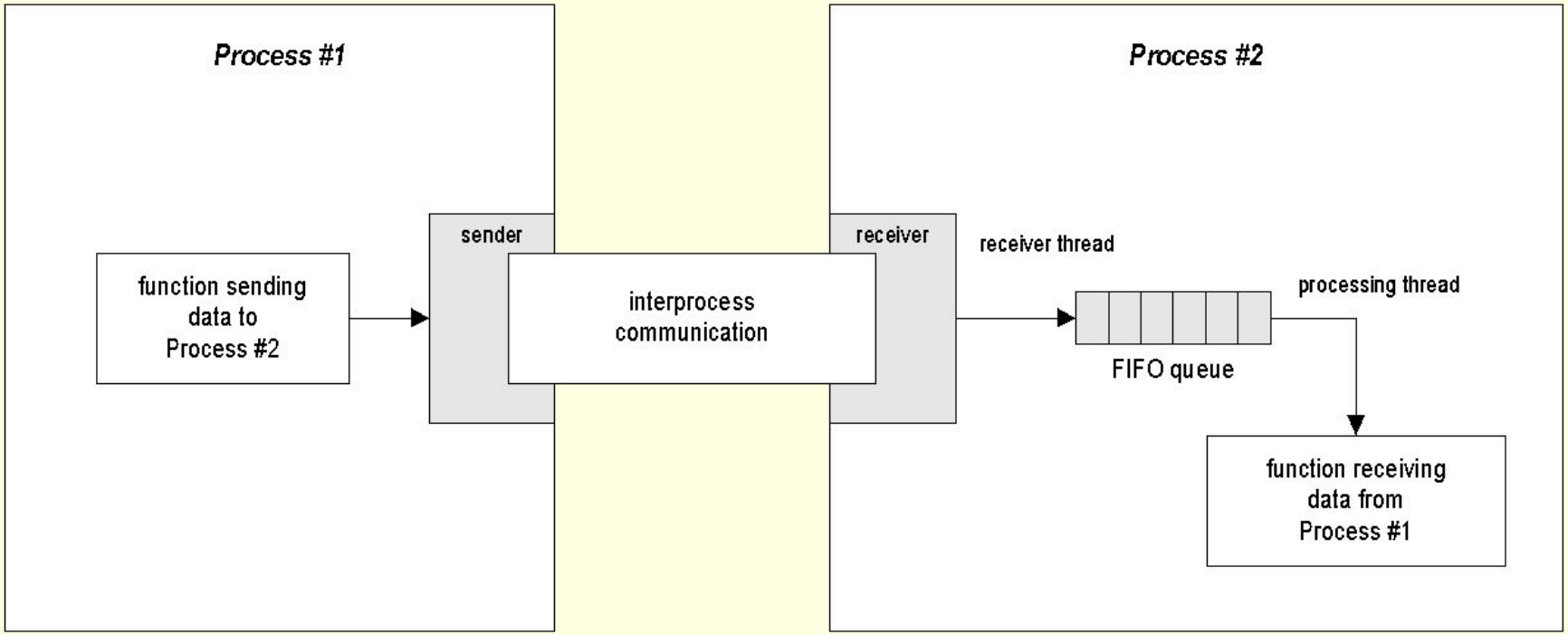
Client/Server Configuration



Socket Data Transfer

- The receiving socket, either client or server, has a buffer that stores bytes of data until the receiver thread reads them.
  - If the receiver buffer is full, the sender thread will block on a send call until the receiver reads some of the data out of the buffer.
    - For this reason, it is a good idea to assign a thread in the receiver to empty the buffer and enqueue the data for a worker thread to digest.
  - If the receiver buffer becomes full during a send, the send request will return having sent less than the requested number of bytes.
  - If the receiving buffer is empty, a read request will block.
  - If the receiving buffer has data, but less than the number of bytes requested by a read, the call will return with the bytes available.

Non-Blocking Communication



Basic .Net Network Objects

- TCPListener
  - TCPListener(port)
  - AcceptTcpClient()
  - AcceptSocket()
  - Start()
  - Stop()
- Socket
  - Send(byte[], size, socketFlags)
  - Receive(byte[], size, socketFlags)
  - Close()
  - ShutDown(SocketShutDown)

More Network Programming Objects

- TCPClient
  - TCPClient()
  - Connect(IPAddress, port)
  - GetStream()
  - Close()
- NetworkStream
  - NetworkStream(Socket)
  - Read(byte[], offset, size)
  - Write(byte[], offset, size)

You read and write using the returned NetworkStream object

Simple Socket Client

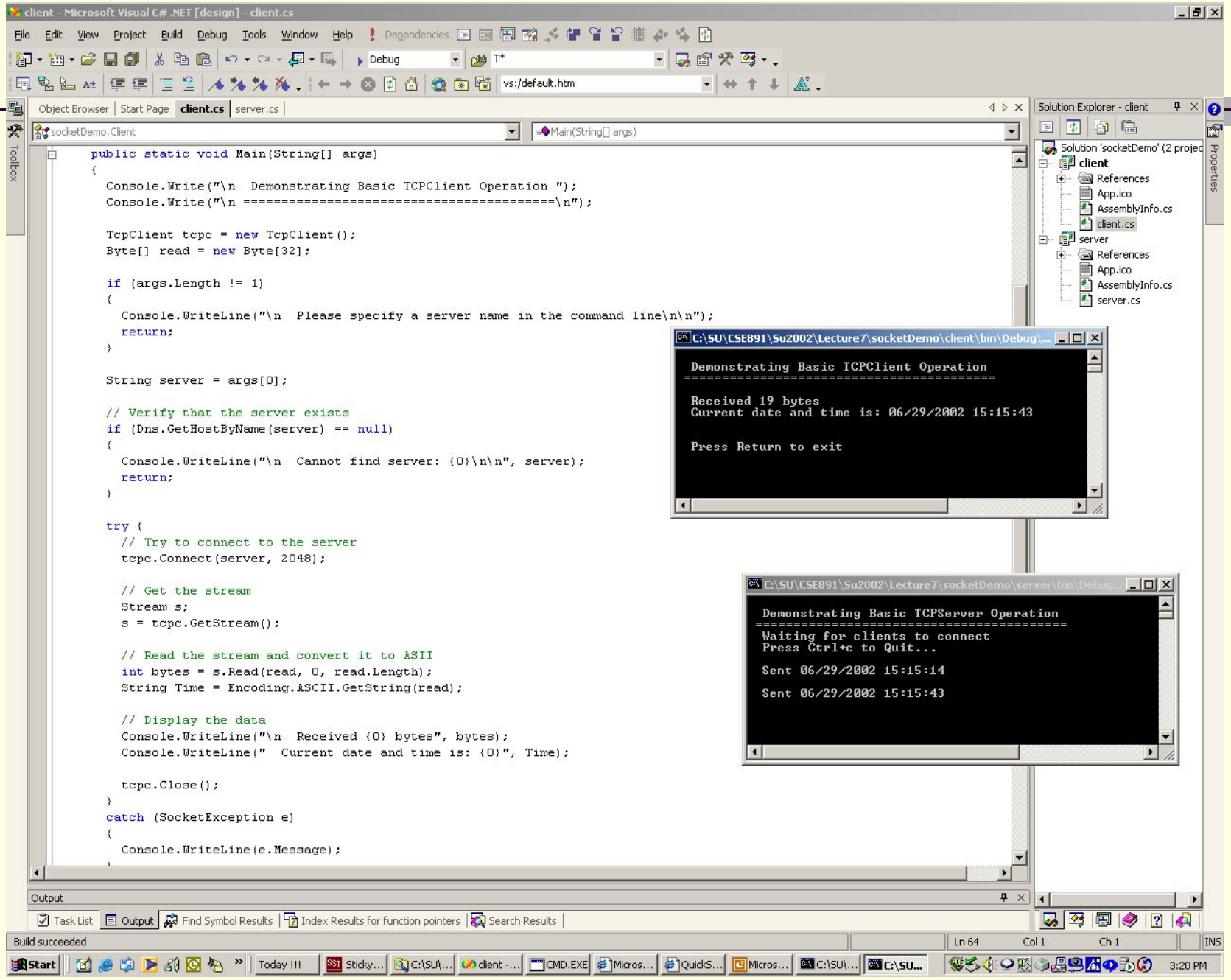
```
   TcpClient tcpc = new TcpClient();
   Byte[] read = new Byte[32];   // read buffer
   String server = args[0];      // server name

// Try to connect to the server
   tcpc.Connect(server, 2048);

// Get a NetworkStream object
   Stream s;
   s = tcpc.GetStream();

// Read the stream and convert it to ASII
   int bytes = s.Read(read, 0, read.Length);
   String Time = Encoding.ASCII.GetString(read);

// Display the data
   Console.WriteLine("\n  Received {0} bytes", bytes);
   Console.WriteLine("  Current date and time is: {0}", Time);

   tcpc.Close();
```

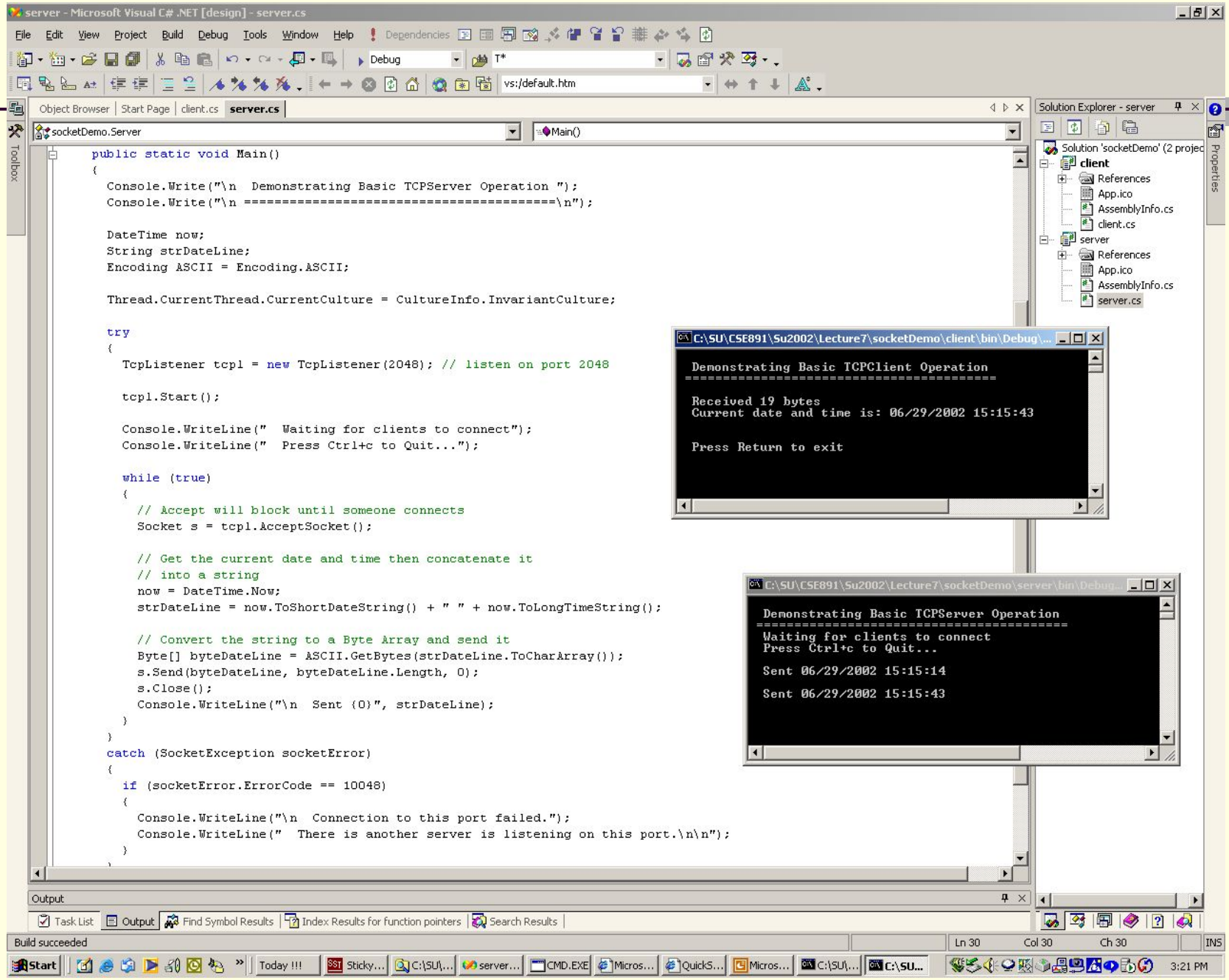
Connects to server with this name

Connects to this server port

Simple Socket Server

```
TcpListener tcpl = new TcpListener(2048); // listen on port 2048

tcpl.Start();

while (true)
{
  // Accept will block until someone connects
  Socket s = tcpl.AcceptSocket();

  // Get current date and time then concatenate it into a string
  now = DateTime.Now;
  strDateLine = now.ToShortDateString()
              + " " + now.ToLongTimeString();

  // Convert the string to a Byte Array and send it
  Byte[] byteDateLine = ASCII.GetBytes(strDateLine.ToCharArray());
  s.Send(byteDateLine, byteDateLine.Length, 0);
  s.Close();
  Console.WriteLine("\n  Sent {0}", strDateLine);
}
```


client - Microsoft Visual C# .NET [design] - client.cs
File Edit View Project Build Debug Tools Window Help
Debug
vs:/default.htm
Object Browser | Start Page | client.cs | server.cs
socketDemo.Client
Main(String[] args)
public static void Main(String[] args)
{
Console.Write("\n Demonstrating Basic TCPClient Operation ");
Console.Write("\n ==\n");
TcpClient tcpc = new TcpClient();
Byte[] read = new Byte[32];
if (args.Length != 1)
{
Console.WriteLine("\n Please specify a server name in the command line\n\n");
return;
}
String server = args[0];
// Verify that the server exists
if (Dns.GetHostByName(server) == null)
{
Console.WriteLine("\n Cannot find server: {0}\n\n", server);
return;
}
try {
// Try to connect to the server
tcpc.Connect(server, 2048);
// Get the stream
Stream s;
s = tcpc.GetStream();
// Read the stream and convert it to ASII
int bytes = s.Read(read, 0, read.Length);
String Time = Encoding.ASCII.GetString(read);
// Display the data
Console.WriteLine("\n Received {0} bytes", bytes);
Console.WriteLine(" Current date and time is: {0}", Time);
tcpc.Close();
}
catch (SocketException e)
{
Console.WriteLine(e.Message);
Solution Explorer - client
Solution 'socketDemo' (2 projec
client
References
App.ico
AssemblyInfo.cs
client.cs
server
References
App.ico
AssemblyInfo.cs
server.cs
C:\SU\CSE891\Su2002\Lecture7\socketDemo\client\bin\Debug\...
Demonstrating Basic TCPClient Operation
Received 19 bytes
Current date and time is: 06/29/2002 15:15:43
Press Return to exit
C:\SU\CSE891\Su2002\Lecture7\socketDemo\server\bin\Debug...
Demonstrating Basic TCPServer Operation
Waiting for clients to connect
Press Ctrl+c to Quit...
Sent 06/29/2002 15:15:14
Sent 06/29/2002 15:15:43
Output
Task List | Output | Find Symbol Results | Index Results for function pointers | Search Results
Build succeeded
Ln 64 Col 1 Ch 1
INS
Start
Today !!!
Sticky...
C:\SU\...
client -...
CMD.EXE
Micros...
QuickS...
Micros...
C:\SU\...
C:\SU...
3:20 PM


server - Microsoft Visual C# .NET [design] - server.cs
File Edit View Project Build Debug Tools Window Help
Debug
vs:/default.htm
Object Browser | Start Page | client.cs | server.cs
socketDemo.Server
Main()
public static void Main()
{
Console.Write("\n Demonstrating Basic TCPServer Operation ");
Console.Write("\n ==\n");
DateTime now;
String strDateLine;
Encoding ASCII = Encoding.ASCII;
Thread.CurrentThread.CurrentCulture = CultureInfo.InvariantCulture;
try
{
TcpListener tcpl = new TcpListener(2048); // listen on port 2048
tcpl.Start();
Console.WriteLine(" Waiting for clients to connect");
Console.WriteLine(" Press Ctrl+c to Quit...");
while (true)
{
// Accept will block until someone connects
Socket s = tcpl.AcceptSocket();
// Get the current date and time then concatenate it
// into a string
now = DateTime.Now;
strDateLine = now.ToShortDateString() + " " + now.ToLongTimeString();
// Convert the string to a Byte Array and send it
Byte[] byteDateLine = ASCII.GetBytes(strDateLine.ToCharArray());
s.Send(byteDateLine, byteDateLine.Length, 0);
s.Close();
Console.WriteLine("\n Sent {0}", strDateLine);
}
}
catch (SocketException socketError)
{
if (socketError.ErrorCode == 10048)
{
Console.WriteLine("\n Connection to this port failed.");
Console.WriteLine(" There is another server is listening on this port.\n\n");
}
Solution Explorer - server
Solution 'socketDemo' (2 projec
client
References
App.ico
AssemblyInfo.cs
client.cs
server
References
App.ico
AssemblyInfo.cs
server.cs
C:\SU\CSE891\Su2002\Lecture7\socketDemo\client\bin\Debug\...
Demonstrating Basic TCPClient Operation
Received 19 bytes
Current date and time is: 06/29/2002 15:15:43
Press Return to exit
C:\SU\CSE891\Su2002\Lecture7\socketDemo\server\bin\Debug...
Demonstrating Basic TCPServer Operation
Waiting for clients to connect
Press Ctrl+c to Quit...
Sent 06/29/2002 15:15:14
Sent 06/29/2002 15:15:43
Output
Task List | Output | Find Symbol Results | Index Results for function pointers | Search Results
Build succeeded
Ln 30 Col 30 Ch 30
INS
Start
Today !!!
Sticky...
C:\SU\...
server...
CMD.EXE
Micros...
QuickS...
Micros...
C:\SU\...
C:\SU...
3:21 PM

Multi-threaded Server

- If we want to support concurrent clients, the server must spawn a thread for each new client.
- C# Thread class makes that fairly simple.
  - Create a class that provides a non-static processing function. This is the code that serves each client.
  - Each time the TCPListener accepts a client it returns a socket. Pass that to the thread when it is constructed, and start the thread.

Define Thread's Processing

```
class threadProc
 {
   private Socket _sock = null;

   public threadProc(Socket sock)
   {
     _sock = sock;
   }
   public void proc()
   {
     for(int i=0; i<20; i++)
     {
       // Get the current date and time then concatenate it
       // into a string
       DateTime now = DateTime.Now;
       string strDateLine = now.ToShortDateString() + " "
                          +  now.ToLongTimeString();

       // Convert the string to a Byte Array and send it
       Byte[] byteDateLine = Encoding.ASCII.GetBytes(strDateLine.ToCharArray());
       _sock.Send(byteDateLine, byteDateLine.Length, 0);
       Console.Write("\n  Sent {0}", strDateLine);
       Thread.Sleep(1000);   // wait for one second just for demo
     }
     string QuitMessage = "Quit";
     Byte[] byteQuit = Encoding.ASCII.GetBytes(QuitMessage.ToCharArray());
     _sock.Send(byteQuit, byteQuit.Length, 0);
     while(_sock.Connected)
       Thread.Sleep(100);
     _sock.Close();
   }
 }
```

Server Spawns Threads to Handle New Clients with threadProc.proc()

```
// listen on port 2048
   TcpListener tcpl = new TcpListener(2048);
   tcpl.Start();

   while (true)
   {
      // Accept will block until someone connects
      Socket s = tcpl.AcceptSocket();

     threadProc tp = new threadProc(s);

     // pass threadProc.proc() function reference to
     // ThreadStart delegate

     Thread t = new Thread(new ThreadStart(tp.proc));
     t.Start();
   }
```

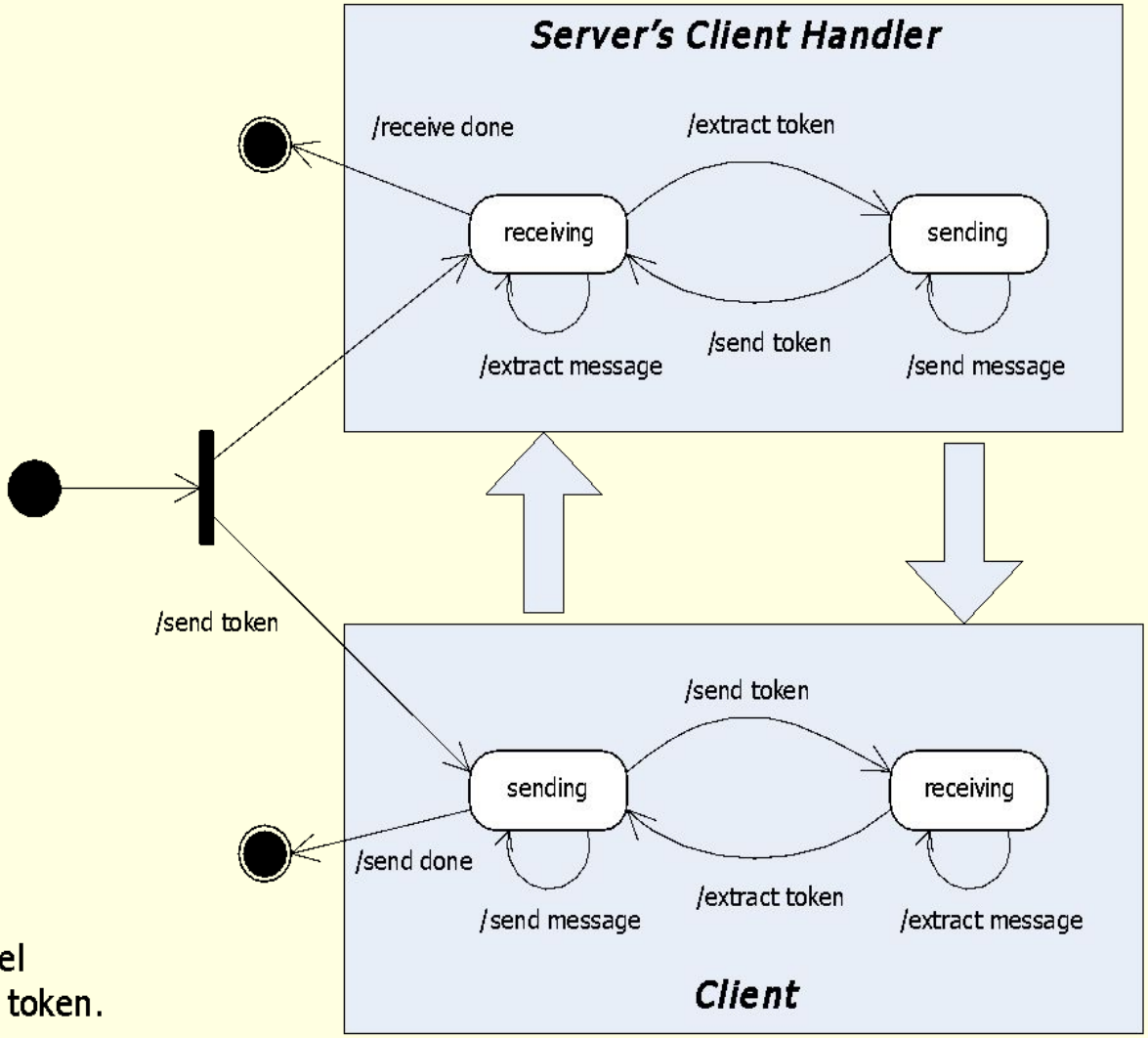
Clients now Wait for Server to Complete

```
// Try to connect to the server
tcpc.Connect(server, 2048);

// Get the NetworkStream object
Stream s;
s = tcpc.GetStream();

while(true)
{
  // Read the stream and convert it to ASII
  int bytes = s.Read(read, 0, read.Length);
  String TSvrMsg = Encoding.ASCII.GetString(read);
  TSrvMsg = TSrvMsg.Remove(bytes,TSrvMsg.Length-bytes);

  // Display the data
  if(TSrvMsg == "Quit")
  {
    Console.Write("\n  Quitting");
    break;
  }
  Console.WriteLine("  Server date and time is: {0}",
TSrvMsg);
}
tcpc.Close();
```

Talk Protocol

- The hardest part of a client/server socket communication design is to control the active participant
  - If single-threaded client and server both talk at the same time, their socket buffers will fill up and they both will block, e.g., deadlock.
  - If they both listen at the same time, again there is deadlock.
  - Often the best approach is to use separate send and receive threads
    - two unilateral communication channels
  - The next slide shows how to safely use bilateral communication.

Bilateral Channel Talk-Listen Protocol



Each connection channel contains one "sending" token.

Message Length

- Another vexing issue is that the receiver may not know how long a sent message is.
  - so the receiver doesn't know how many bytes to pull from the stream to compose a message.
  - Often, the communication design will arrange to use message delimiters, fixed length messages, or message headers that carry the message length as a parameter.

Message Framing

- There are three solutions to this problem:
  - Use fixed length messages – rarely useful
  - Use fixed length message headers
    - Encode message body length in header
    - Reader pulls header, parses to find length of rest of message and pulls it.
  - Use message termination sentinals
    - <msg>body of message</msg>
    - Reader reads a character at a time out of channel
    - Adds character to message
    - Scans message from back looking for </msg> to conclude message extraction.

They're Everywhere

- Virtually every network and internet communication method uses sockets, often in a way that is invisible to an application designer.
  - Browser/server
  - ftp
  - SOAP
  - Network applications

Sockets

The End