using IXmlDB;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace ServerDemo

{

class Program

{

// 当前通信的socket和客户端socket

static Socket currentSocket;

// 风扇状态

static string fan\_state = "关";

// 当前温度

static string temp = "";

static void Main(string[] args)

{

// 启动socket服务器

socketServer();

// 启动http服务器

httpServer();

// 记录温度

writeTemp();

}

// socket服务器

static void socketServer()

{

// 创建一个在服务器端负责监听IP地址和端口号的Socket

Socket socketWatch = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

// IP

IPAddress ip = IPAddress.Any;

// 端口

IPEndPoint point = new IPEndPoint(ip, 5000);

// 开始监听

socketWatch.Bind(point);

Console.WriteLine("Socket 服务器启动成功! " + point.ToString());

// 10毫秒监听一次

socketWatch.Listen(10);

// 开启一个新线程

// 用于等待客户端连接

Thread th = new Thread(socketListen);

th.Start(socketWatch);

}

// 监听

static void socketListen(object o)

{

Socket socketWatch = o as Socket;

while (true)

{

try

{

// 等待连接

Socket tempSocket = socketWatch.Accept();

if (currentSocket != null)

{

// 关闭当前连接

currentSocket.Shutdown(SocketShutdown.Both);

currentSocket.Close();

}

currentSocket = tempSocket;

Console.WriteLine(currentSocket.RemoteEndPoint.ToString() + ": 连接成功!");

Thread th = new Thread(socketRevice);

th.Start();

}

catch (Exception)

{

// 不处理，有的异常是不影响程序运行的

// 但是影响体验

}

}

}

// 接收消息

static void socketRevice()

{

while (true)

{

try

{

byte[] buffer = new byte[10];

// 等待接收数据

int r = currentSocket.Receive(buffer);

string rec = Encoding.ASCII.GetString(buffer, 0, r);

Console.WriteLine(currentSocket.RemoteEndPoint.ToString() + ": " + rec);

if (r == 0)

{

Console.WriteLine("关闭连接: " + currentSocket.RemoteEndPoint.ToString());

// 关闭当前连接

currentSocket.Shutdown(SocketShutdown.Both);

currentSocket.Close();

currentSocket = null;

break;

}

if ((char)buffer[0] == '0')

{

// 接收风扇状态

fan\_state = (char)buffer[1] == '1' ? "开" : "关";

Console.WriteLine("设置风扇状态为：" + (char)buffer[1]);

}

else if ((char)buffer[0] == '1')

{

// 接收温度数据

temp = Encoding.ASCII.GetString(buffer, 1, r - 1);

Console.WriteLine("接收温度：" + temp);

}

else

{

Console.WriteLine("错误数据! buffer[0]: "+ (char)buffer[0]);

}

}

catch (Exception)

{

//

}

}

}

// Http服务器

static void httpServer()

{

// 创建一个侦听器

HttpListener httplistener = new HttpListener();

// 配置身份验证

httplistener.AuthenticationSchemes = AuthenticationSchemes.Anonymous;

// 绑定侦听的url和端口

httplistener.Prefixes.Add("http://zigbee.ydath.cn:80/");

// 测试用

//httplistener.Prefixes.Add("http://localhost:8080/");

// 开始侦听

httplistener.Start();

new Thread(new ThreadStart(delegate

{

while (true)

{

// 获得http上下文

HttpListenerContext httpListenerContext = httplistener.GetContext();

// 获取请求全路径

string requst = httpListenerContext.Request.Url.ToString();

string req = requst.Contains("/") ? requst.Substring(requst.LastIndexOf('/') + 1) : "index.html";

req = req == "" ? "index.html" : req;

// 设置响应码

httpListenerContext.Response.StatusCode = 200;

if (req == "index.html")

{

// 返回首页数据

using (StreamWriter writer = new StreamWriter(httpListenerContext.Response.OutputStream))

{

// 返回数据

writer.Write(File.ReadAllText("html/index.html", Encoding.UTF8));

Console.WriteLine("200 " + req + " RequestUrl: " + httpListenerContext.Request.Url);

}

}

else if (req.Contains("?state"))

{

// 返回状态信息 使用json格式

using (StreamWriter writer = new StreamWriter(httpListenerContext.Response.OutputStream))

{

// 返回数据

writer.Write("{\"temp\":\"" + (temp == "" ? "0" : temp) + "\", \"state\":\"" + fan\_state + "\"}");

Console.WriteLine("200 " + req + " RequestUrl: " + httpListenerContext.Request.Url);

}

}

else if (req.Contains("?fan\_on"))

{

Console.WriteLine("200 " + req + " RequestUrl: " + httpListenerContext.Request.Url);

// 开关灯命令

if (currentSocket != null)

{

currentSocket.Send(new byte[] { 1 });

}

fan\_state = "开";

Console.WriteLine("转发开灯命令成功!");

using (StreamWriter writer = new StreamWriter(httpListenerContext.Response.OutputStream))

{

// 返回数据

writer.Write("");

}

}

else if (req.Contains("?fan\_off"))

{

Console.WriteLine("200 " + req + " RequestUrl: " + httpListenerContext.Request.Url);

// 开关灯命令

if (currentSocket != null)

{

currentSocket.Send(new byte[] { 0 });

}

fan\_state = "关";

Console.WriteLine("转发关灯命令成功!");

using (StreamWriter writer = new StreamWriter(httpListenerContext.Response.OutputStream))

{

// 返回数据

writer.Write("");

}

}

else

{

// 错误的参数, 不处理

Console.WriteLine("web服务器请求参数错误!");

using (StreamWriter writer = new StreamWriter(httpListenerContext.Response.OutputStream))

{

// 返回数据

writer.Write("");

}

}

}

})).Start();

Console.WriteLine("web服务器启动成功!");

}

// 写温度

static void writeTemp()

{

new Thread(new ThreadStart(delegate

{

// 向XML文件中写入温度数据

while (true)

{

if (temp != "")

{

DbContext db = new DbContext();

db.TempLogs.Add(new TempLog { Temp = temp });

db.TempLogs.SaveChanges();

}

// 每五秒写一次

Thread.Sleep(5000);

}

})).Start();

Console.WriteLine("温度记录模块启动成功!");

}

}

// 温度数据模型

class TempLog

{

// Id

public string Id { get; set; }

// 温度

public string Temp { get; set; }

}

class DbContext

{

// 连接User.xml数据文件，若不存在，会自动创建

// 以日期分割成文件夹，以小时分割为文件

public XmlDbSet<TempLog> TempLogs { get; set; } = new XmlDbSet<TempLog>(@"templog\" + DateTime.Now.Year.ToString() + "-" + DateTime.Now.Month.ToString() + "-" + DateTime.Now.Day.ToString() + @"\" + DateTime.Now.Hour.ToString() + ".xml", "TempLog", "TempLogs");

}

}