**LWIP Server (MCU) C** **Net.Sockets Client (PC) C#**  
  
**LWIP\_INIT**() **string IP = “192.168.100.1”  
IP4\_ADDR**(&IP, 192, 168, 100, 1); **IPEndPoint ep = new IPEndPoint(IP, port);**   
**struct tcp\_pcb \*pcb; 1) Socket client = new Socket(  
 ep.AddressFamily  
1)** pcb **= tcp\_new**(); **SocketType.Stream  
 tcp\_bind**(pcb, &IP, port); **ProtocolType.Tcp);  
 tcp\_listen**(pcb); **tcp\_accept**(pcb**, cb\_accept);**  **2) client.Connect(ep);**   
**2)** **cb\_accept(arg, newpcb, err)** {   
struct tcp\_server\_struct \*es;  
 **LWIP\_UNUSED\_ARG**(arg)  
 **LWIP\_UNUSED\_ARG**(err)  
 **tcp\_set\_prio**(newpcb, TCP\_PRIO\_MIN)  
  
 es = (struct tcp\_server\_struct \*)mem\_malloc(  
 sizeof(struct tcp\_server\_struct));  
  
 es->state = ES\_ACCEPTED;  
 es->pcb = newpcb;  
 es->p = NULL;  
  
 **tcp\_arg**(newpcb, es);  
 **tcp\_recv**(newpcb, **cb\_recved**);  
 **tcp\_err**(newpcb, **cb\_error**);  
 **tcp\_poll**(newpcb, **cb\_poll**, 1);   
  
 ret\_err = ERR\_OK; byte [] msg = Encoding.ASCII.GetBytes(str)  
 return ret\_err; **3) int sentCnt = client.Send()**  
 }   
 **3) cb\_recved(arg, tpcb, p, err)** {  struct tcp\_server\_struct \*es;  
 err\_t ret\_err;  
  
 **LWIP\_ASSERT**("arg != NULL",arg != NULL);  
 es = (struct tcp\_server\_struct \*)arg;

PSH, ACK

ACK

SYN, ACK

SYN

if (p == NULL)  
 {  
 es->state = ES\_CLOSING;  
 if(es->p == NULL)  
 { **5) client.Shutdown(**  
 **server\_close**(tpcb, es);  **SocketShutdown.Both);**  
 }  
 else  
 {  
 **tcp\_sent**(tpcb, **cb\_sent**);  
 **server\_send**(tpcb, es);  
 }  
 ret\_err = ERR\_OK;  
 }  
 else if(err != ERR\_OK)  
 {  
 if (p != NULL)  
 {  
 es->p = NULL;  
 **pbuf\_free**(p);  
 }  
 ret\_err = err;  
 }  
 else if(es->state == ES\_ACCEPTED)  
 {  
 es->state = ES\_RECEIVED;  
 es->p = p;  
 **tcp\_sent**(tpcb, **cb\_sent**);  
 **server\_send**(tpcb, es);  
 ret\_err = ERR\_OK;  
 }  
 else if (es->state == ES\_RECEIVED)  
 {  
 if(es->p == NULL)  
 {  
 es->p = p;  
 **server\_send**(tpcb, es);  
 }  
 else  
 {  
 struct pbuf \*ptr;  
 ptr = es->p;  
 **pbuf\_chain**(ptr,p);  
 }  
 ret\_err = ERR\_OK;  
 }

FIN, ACK

else  
 {  
 **tcp\_recved**(tpcb, p->tot\_len);  
 es->p = NULL;  
 **pbuf\_free**(p);  
 ret\_err = ERR\_OK;  
 }  
 return ret\_err;  
 }  
   
 **server\_send**(tpcb, es)   
 {  
 struct pbuf \*ptr;  
 err\_t wr\_err = ERR\_OK;  
  
 **while** ((wr\_err == ERR\_OK) &&  
 (es->p != NULL) &&  
 (es->p->len <= **tcp\_sndbuf**(tpcb)))  
 {  
 ptr = es->p;  
 wr\_err = **tcp\_write**(tpcb,   
 ptr->payload,  
 ptr->len, 1);

if (wr\_err == ERR\_OK)  
 {  
 u16\_t plen;  
 plen = ptr->len;  
 char \*pReceived = (char\*)**mem\_malloc**(  
 (size\_t)(plen + 1));  
  
 es->p = ptr->next;  
 if(es->p != NULL)  
 {  
 **pbuf\_ref**(es->p);  
 }  
 **pbuf\_free**(ptr); **4) int recvdCnt = client.Receive(bytes);**  
 **tcp\_recved**(tpcb, plen);  
 }  
 else if(wr\_err == ERR\_MEM)  
 {  
 es->p = ptr;  
 }  
 else  
 {}  
 }

PSH, ACK

}

**cb\_sent**(arg, tpcb, len)  
 {  
 struct tcp\_server\_struct \*es;  
 **LWIP\_UNUSED\_ARG**(len);  
 es = (struct tcp\_server\_struct \*)arg;  
  
 if(es->p != NULL)  
 {  
 **server\_send**(tpcb, es);  
 }  
 else  
 {  
 if(es->state == ES\_CLOSING)  
 **server\_close**(tpcb, es);  
 }

return ERR\_OK;

}  
  
 **server\_poll**(arg, tpcb)  
 {  
 err\_t ret\_err;  
 struct tcp\_server\_struct \*es;  
 es = (struct tcp\_server\_struct \*)arg;  
 if (es != NULL)  
 {  
 if (es->p != NULL)  
 {  
 **server\_send**(tpcb, es);  
 }  
 else  
 {  
 if(es->state == ES\_CLOSING)  
 {  
 **server \_close**(tpcb, es);  
 }  
 }  
 ret\_err = ERR\_OK;  
 }  
 else  
 {  
 **tcp\_abort**(tpcb);  
 ret\_err = ERR\_ABRT;  
 }  
 return ret\_err;  
 }

**server\_close**(tpcb, es)  
 {  
 **tcp\_arg**(tpcb, NULL);  
 **tcp\_sent**(tpcb, NULL);  
 **tcp\_recv**(tpcb, NULL);  
 **tcp\_err**(tpcb, NULL);  
 **tcp\_poll**(tpcb, NULL, 0);  
  
 if (es != NULL)  
 {  
 **mem\_free**(es);  
 }   
 **tcp\_close**(tpcb);   
 }

FIN, ACK

ACK

**cb\_error**(arg, err)  
 {  
 struct tcp\_server\_struct \*es;  
 **LWIP\_UNUSED\_ARG**(err);  
  
 es = (struct tcp\_server\_struct \*)arg;  
  
 if (es != NULL)  
 {  
 **mem\_free**(es);  
 }  
 }