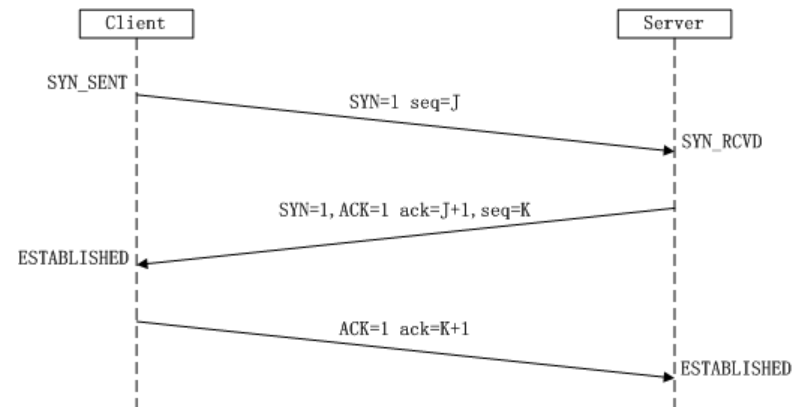
# 1 TCP与UDP协议理论

## 1.1 TCP的三次握手与四次挥手

三次握手



第一次握手：Client将标志位SYN置为1，随机产生一个值seq=J，并将该数据包发送给Server，Client进入SYN\_SENT状态，等待Server确认。

第二次握手：Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack=J+1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN\_RCVD状态。

第三次握手：Client收到确认后，检查ack是否为J+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给Server，Server检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了。

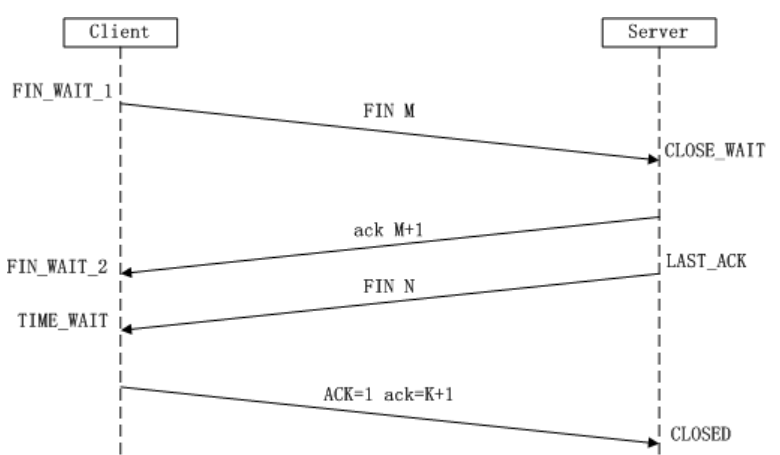
简单来说，就是：

1）：建立连接时，客户端发送SYN包（SYN=i）到服务器，并进入到SYN-SEND状态，等待服务器确认

2）：服务器收到SYN包，必须确认客户的SYN（ack=i+1）,同时自己也发送一个SYN包（SYN=k）,即SYN+ACK包，此时服务器进入SYN-RECV状态

3）：客户端收到服务器的SYN+ACK包，向服务器发送确认报ACK（ack=k+1）,此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手，客户端与服务器开始传送数据。

四次挥手



由于TCP连接时全双工的，因此，每个方向都必须要单独进行关闭，这一原则是当一方完成数据发送任务后，发送一个FIN来终止这一方向的连接，收到一个FIN只是意味着这一方向上没有数据流动了，即不会再收到数据了，但是在这个TCP连接上仍然能够发送数据，直到这一方向也发送了FIN。首先进行关闭的一方将执行主动关闭，而另一方则执行被动关闭，上图描述的即是如此。

第一次挥手：Client发送一个FIN，用来关闭Client到Server的数据传送，Client进入FIN\_WAIT\_1状态。

第二次挥手：Server收到FIN后，发送一个ACK给Client，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），Server进入CLOSE\_WAIT状态。

第三次挥手：Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入LAST\_ACK状态。

第四次挥手：Client收到FIN后，Client进入TIME\_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，Server进入CLOSED状态，完成四次挥手。

为什么建立连接是三次握手，而关闭连接却是四次挥手呢？

这是因为服务端在LISTEN状态下，收到建立连接请求的SYN报文后，把ACK和SYN放在一个报文里发送给客户端。而关闭连接时，当收到对方的FIN报文时，仅仅表示对方不再发送数据了但是还能接收数据，己方也未必全部数据都发送给对方了，所以己方可以立即close，也可以发送一些数据给对方后，再发送FIN报文给对方来表示同意现在关闭连接，因此，己方ACK和FIN一般都会分开发送。

## 1.2 TCP与UDP区别总结：

（1）TCP面向连接（如打电话要先拨号建立连接）; UDP是无连接的，即发送数据之前不需要建立连接

（2）TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付。Tcp通过校验和，重传控制，序号标识，滑动窗口、确认应答实现可靠传输。如丢包时的重发控制，还可以对次序乱掉的分包进行顺序控制。

（3）UDP具有较好的实时性，工作效率比TCP高，适用于对高速传输和实时性有较高的通信或广播通信。

（4）每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信。

（5）TCP对系统资源要求较多，UDP对系统资源要求较少。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 属性 | 建立连接 | 可靠性 | 实时性 | 资源消耗 | 传输方式 | 典型应用 |
| TCP | 是 | 高 | 低 | 多 | 点对点 | HTTP\FTP\POP3等 |
| UDP | 否 | 低 | 高 | 少 | 多对多 | NTP\DHCP\DNS\ECHO等 |

## 1.3 为什么UDP有时比TCP更有优势?

UDP以其简单、传输快的优势，在越来越多场景下取代了TCP,如实时游戏。

（1）网速的提升给UDP的稳定性提供可靠网络保障，丢包率很低，如果使用应用层重传，能够确保传输的可靠性。

（2）TCP为了实现网络通信的可靠性，使用了复杂的拥塞控制算法，建立了繁琐的握手过程，由于TCP内置的系统协议栈中，极难对其进行改进。

采用TCP，一旦发生丢包，TCP会将后续的包缓存起来，等前面的包重传并接收到后再继续发送，延时会越来越大，基于UDP对实时性要求较为严格的情况下，采用自定义重传机制，能够把丢包产生的延迟降到最低，尽量减少网络问题对游戏性造成影响。

## 1.4 UDP和TCP编程步骤也有些不同，如下：

**TCP编程的服务器端一般步骤是：**

（1）创建一个socket，用函数socket()；

（2）设置socket属性，用函数setsockopt(); \* 可选

（3）绑定IP地址、端口等信息到socket上，用函数bind();

（4）开启监听，用函数listen()；

（5）接收客户端上来的连接，用函数accept()；

（6）收发数据，用函数send()和recv()，或者read()和write();

（7）关闭网络连接；

（8）关闭监听；

SOCK\_STREAM这种的特点是面向连接的，即每次收发数据之前必须通过connect建立连接，而SOCK\_DGRAM这种是User Datagram Protocol协议的网络通讯，它是无连接的，不可靠的。

**TCP编程的客户端一般步骤是：**

（1）创建一个socket，用函数socket()；

（2）设置socket属性，用函数setsockopt();\* 可选

（3）绑定IP地址、端口等信息到socket上，用函数bind();\* 可选

（4）设置要连接的对方的IP地址和端口等属性；

（5）连接服务器，用函数connect()；

（6）收发数据，用函数send()和recv()，或者read()和write();

（7）关闭网络连接；

**UDP编程的服务器端一般步骤是：**

（1）创建一个socket，用函数socket()；

（2）设置socket属性，用函数setsockopt();\* 可选

（3）绑定IP地址、端口等信息到socket上，用函数bind();

（4）循环接收数据，用函数recvfrom();

（5）关闭网络连接；

**UDP编程的客户端一般步骤是：**

（1）创建一个socket，用函数socket()；

（2）设置socket属性，用函数setsockopt();\* 可选

（3）绑定IP地址、端口等信息到socket上，用函数bind();\* 可选

（4）设置对方的IP地址和端口等属性;

（5）发送数据，用函数sendto();

（6）关闭网络连接；

## 1.5 本实验基本框架

初始化客户端IP

远程服务器IP与端口

创建套接字

收发数据

关闭套接字

本地服务器地址绑定

初始化服务器IP

创建套接字

本地服务器IP与端口

收发数据

**UDP协议：**

初始化客户端IP

远程服务器IP与端口

创建套接字

请求连接远程服务器

收发数据

关闭套接字

开启监听

本地服务器地址绑定

初始化服务器IP

创建套接字

本地服务器IP与端口

等待客户端接入

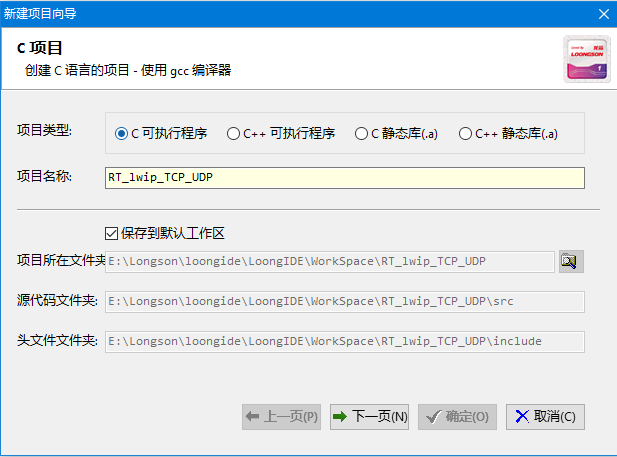
收发数据

被动断开连接

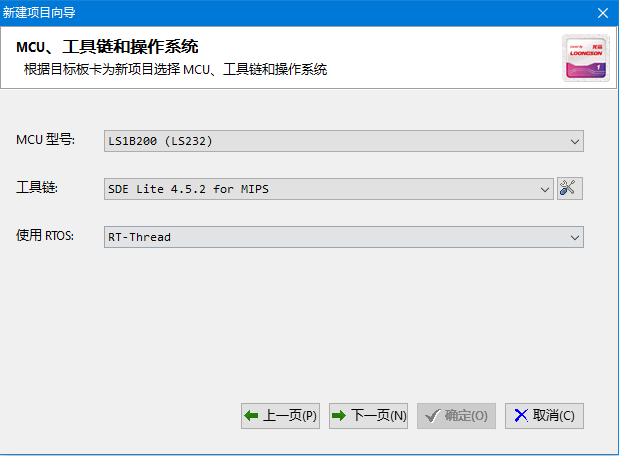
**TCP协议：**

# 2 创建工程

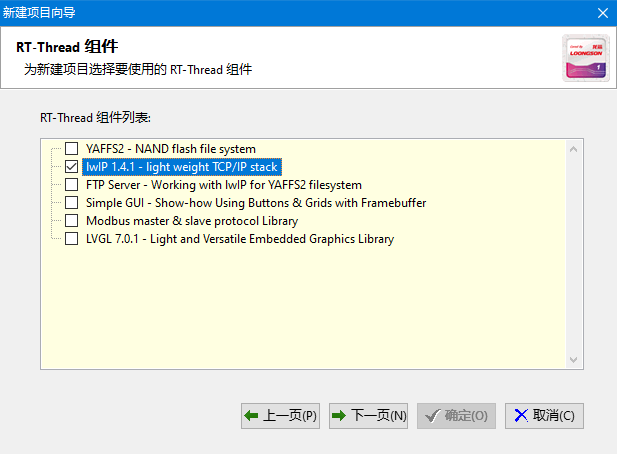
（1）打开LoongIDE新建一个项目。



（2）配置MCU型号、工具链以及操作系统。



（3）勾选lwip组件。



（4）修改main文件：

①将#include "bsp.h"下方宏定义的灰色区域替换为：

|  |
| --- |
| #ifdef BSP\_USE\_GMAC0  static unsigned char gmac0\_ip[4] = {192, 168, 1, 123};  extern void ls1x\_initialize\_lwip(unsigned char \*ip0, unsigned char \*ip1);  #endif |

②删除原有样例线程“demo\_thread”。替换为：

|  |
| --- |
| #include "lwip\_test.h"  int main(int argc, char\*\* argv)  {  rt\_kprintf("\r\nWelcome to RT-Thread.\r\n\r\n");  #ifdef BSP\_USE\_GMAC0  {  ls1x\_initialize\_lwip(gmac0\_ip, NULL); // 初始化本地IP地址，参数（GMAC0，GAC1），参数为空时用默认地址  #if (TEST\_TCP\_SERVER)  {  extern void tcp\_server\_init(void);  tcp\_server\_init(); // 开启tcp服务器  }  #endif  #if (TEST\_TCP\_CLIENT)  {  extern void tcp\_client\_init(void);  tcp\_client\_init(); // 开启tcp客户端  }  #endif    #if (TEST\_UDP\_SERVER)  {  extern void udp\_server\_init(void);  udp\_server\_init(); // 开启udp服务器  }  #endif  #if (TEST\_UDP\_CLIENT)  {  extern void udp\_client\_init(void);  udp\_client\_init(); // 开启udp客户端  }  #endif  }  #endif  return 0;    }  /\*  \* @@ End  \*/ |

（5）添加测试头文件“lwip\_test.h”。

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 网络状态  \*/  #define LWIP\_CONNECTED 0x0001  #define LWIP\_SEND\_DATA 0x0002  #define LWIP\_NEW\_DATA 0x0004  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 测试TCP  \*/  #define TCP\_LOCAL\_PORT 9060 // 本地端口  #define TCP\_REMOTE\_PORT 9061 // 远程端口  extern char \*local\_IP; // "192.168.1.123"  extern char \*remote\_IP; // "192.168.1.111"  /\*  \* TCP 状态 - 和网络状态合并  \*/  #define MYTCP\_STATE\_NONE 0  #define MYTCP\_STATE\_RECVDATA 1 // 接收到了数据  #define MYTCP\_STATE\_CLOSED 2 // 连接关闭  typedef struct mytcp\_state  {  unsigned int state;  } mytcp\_state\_t;  /\*  \* 测试 TCP server  \*/  #define TEST\_TCP\_SERVER 0  #if (TEST\_TCP\_SERVER)  #define TCP\_SERVER\_BUFSIZE 256 // 数据缓冲区大小  extern void tcp\_server\_init(void);  #endif  /\*  \* 测试 TCP client  \*/  #define TEST\_TCP\_CLIENT 0  #if (TEST\_TCP\_CLIENT)  #define TCP\_CLIENT\_BUFSIZE 256 // 数据缓冲区大小  extern void tcp\_client\_init(void);  #endif  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 测试UDP  \*/  #define UDP\_LOCAL\_PORT 9062 // 本地端口  #define UDP\_REMOTE\_PORT 9063 // 远程端口  /\*  \* 测试 UDP server. RT-Thread OK 2021.7.16  \*/  #define TEST\_UDP\_SERVER 1  #if (TEST\_UDP\_SERVER)  #define UDP\_SERVER\_BUFSIZE 256 // 数据缓冲区大小  extern void udp\_server\_init(void);  #endif  /\*  \* 测试 UDP client. RT-Thread OK 2021.7.16  \*/  #define TEST\_UDP\_CLIENT 0  #if (TEST\_UDP\_CLIENT)  #define UDP\_CLIENT\_BUFSIZE 256 // 数据缓冲区大小  extern void udp\_client\_init(void);  #endif |

（6）添加测试源文件“lwip\_test.c”。

|  |
| --- |
| #include "lwip\_test.h"  char \*local\_IP = "192.168.1.123"; // 本地IP（Ls1B）  char \*remote\_IP = "192.168.1.111"; // 远程IP（PC机） |

（7）添加TCP服务器源文件“tcp\_server.c”。

|  |
| --- |
| #include "lwip\_test.h"  #if (TEST\_TCP\_SERVER)  #include "lwip/sockets.h"  #include "lwip/err.h"  #include "lwip/sys.h"  static char data\_buf[TCP\_SERVER\_BUFSIZE] = "hello, I'm tcp server!\n";  static void tcp\_server\_thread(void \*arg)  {  struct sockaddr\_in local\_addr; // 定义远程服务器地址  int sock\_fd, err; // 定义套接字，错误代码  sock\_fd = socket(AF\_INET, SOCK\_STREAM, 6); // 创建套接字（级别、类型、协议）  if (sock\_fd == -1)  {  printk("failed to create sock\_fd!\n");  return;  }  else  printk("succeed to create sock\_fd!\n");  memset(&local\_addr, 0, sizeof(local\_addr)); // 为本地服务器地址开辟空间  local\_addr.sin\_family = AF\_INET; // 本地服务器IP有两个部分组成  local\_addr.sin\_addr.s\_addr = inet\_addr(local\_IP); // 本地服务器IP地址192.168.1.123  local\_addr.sin\_port = htons(TCP\_LOCAL\_PORT); // 本地服务器端口号9060  local\_addr.sin\_len = sizeof(local\_addr); // 本地服务器地址占用的存储空间大小  err = bind(sock\_fd, (struct sockaddr \*)&local\_addr, sizeof(struct sockaddr)); // 本地服务器地址绑定  if (err != ERR\_OK)  {  closesocket(sock\_fd); // 绑定失败，关闭套接字  printk("failed to bind ip!\n");  return;  }  else  printk("succeed to bind ip!\n");  err = listen(sock\_fd, 3); // 开启监听  if (err != ERR\_OK)  {  closesocket(sock\_fd);  printk("failed to listen!\n");  return;  }  else  printk("succeed to listen!\n");  /\*  \* loop first.  \*/  while (1)  {  int client\_fd;  struct sockaddr\_in client\_addr;  int addrlen = sizeof(client\_addr);  printk("waiting for client access!\n");  client\_fd = accept(sock\_fd, (struct sockaddr\*)&client\_addr, (socklen\_t)&addrlen); // 客户端接入  if (client\_fd > 0)  {  printk("client incoming succeed!\r\n");    /\*  \* loop second.  \*/  for (;;) /\* 循环接收并自动转发，直至客户端主动断开连接 \*/  {  memset(data\_buf, 0, TCP\_SERVER\_BUFSIZE); // 清除数据缓存区  err = recv(client\_fd, data\_buf, TCP\_SERVER\_BUFSIZE, 0); // 接收数据  if (err > 0) // 正常接收数据并转发  {  printk("RECEIVE: %s \n", data\_buf);  send(client\_fd, data\_buf, err, 0);  printk(" SEND: %s \n", data\_buf);  }  else if (err == ERR\_CLSD) // 客户端连接中断  {  closesocket(client\_fd); // 关闭套接字  printk("client disconnected.\r\n");  break;  }  else if (err <= 0) // 客户端主动断开连接  {  closesocket(client\_fd);  printk("disconnect client succeed!\r\n");  break;  }  }  }  delay\_ms(200);  }  /\*  \* NEVER GO HERE!  \*/  closesocket(sock\_fd); // 关闭套接字  printk("tcp\_server\_thread stop!\r\n"); // 结束  }  void tcp\_server\_init(void) // 开启线程  {  sys\_thread\_new("tcp\_server",  tcp\_server\_thread,  NULL,  DEFAULT\_THREAD\_STACKSIZE,  DEFAULT\_THREAD\_PRIO + 1);  }  #endif |

（8）添加TCP客户端源文件“tcp\_client.c”

|  |
| --- |
| #include "lwip\_test.h"  #if (TEST\_TCP\_CLIENT)  #include "lwip/sockets.h"  #include "lwip/err.h"  #include "lwip/sys.h"  static char data\_buf[TCP\_CLIENT\_BUFSIZE] = "hello, you are connected!\n";  static void tcp\_client\_thread(void \*arg)  {  struct sockaddr\_in remote\_addr; // 定义远程服务器地址  int sock\_fd, err, count=0; // 定义套接字，错误代码，计数  sock\_fd = socket(AF\_INET, SOCK\_STREAM, 0); // 创建套接字（级别、类型、协议）  if (sock\_fd == -1)  {  printk("failed to create sock\_fd!\n");  return;  }  else  printk("succeed to create sock\_fd!\n");  memset(&remote\_addr, 0, sizeof(remote\_addr)); // 为远程服务器地址开辟空间  remote\_addr.sin\_family = AF\_INET; // 远程服务器IP有两个部分组成  remote\_addr.sin\_addr.s\_addr = inet\_addr(remote\_IP); // 远程服务器IP地址192.168.1.111  remote\_addr.sin\_port = htons(TCP\_REMOTE\_PORT); // 远程服务器端口号9061  err = connect(sock\_fd, (struct sockaddr \*)&remote\_addr, sizeof(struct sockaddr)); // 连接远程服务器（套接字、地址）  if (err != ERR\_OK)  {  closesocket(sock\_fd);  printk("failed to connect to server!\n");  return;  }  else  printk("succeed to connect to server!\n");  while (1)  {  int rdbytes = 0;  unsigned int ticks = get\_clock\_ticks(); // 获取时间值    memset(data\_buf, 0, TCP\_CLIENT\_BUFSIZE); // 清除数据缓存区  snprintf(data\_buf, 99, "client ticks = %i.\n", ticks); // 加上换行"\n": 接收端需要【将字符串放在数据缓存区】  if (send(sock\_fd, data\_buf, strlen(data\_buf), 0) <= 0) // 发送数据缓存区数据  {  delay\_ms(1000);  continue;  }  else  {  printk("SEND: %s", data\_buf);  }  memset(data\_buf, 0, TCP\_CLIENT\_BUFSIZE); // 清除数据缓存区  rdbytes = recv(sock\_fd, data\_buf, TCP\_CLIENT\_BUFSIZE, 0); // 接收服务器发来的数据  if (rdbytes > 0)  {  printk(" RECEIVE: %s \n", data\_buf); // 打印服务器发来的数据  }    delay\_ms(100);  if (++count >= 5) /\* 结束标志 5次收发 \*/  break;  }    closesocket(sock\_fd); // 关闭套接字  printk("tcp\_client\_thread stop!\r\n"); // 结束  }  void tcp\_client\_init(void) // 开启线程  {  sys\_thread\_new("tcp\_client",  tcp\_client\_thread,  NULL,  DEFAULT\_THREAD\_STACKSIZE,  DEFAULT\_THREAD\_PRIO + 1);  }  #endif |

（9）添加UDP服务器源文件”udp\_server.c”

|  |
| --- |
| #include "lwip\_test.h"  #if (TEST\_UDP\_SERVER)  #include "lwip/sockets.h"  #include "lwip/err.h"  #include "lwip/sys.h"  static char data\_buf[UDP\_SERVER\_BUFSIZE];  static void udp\_server\_thread(void \*arg)  {  struct sockaddr\_in local\_addr; // 定义远程服务器地址  int sock\_fd,err; // 定义套接字，错误代码  sock\_fd = socket(AF\_INET, SOCK\_DGRAM, 0); // 创建套接字（级别、类型、协议）  if (sock\_fd == -1)  {  printk("failed to create sock\_fd!\n");  return;  }  else  printk("succeed to create sock\_fd!\n");  memset(&local\_addr, 0, sizeof(local\_addr)); // 为本地服务器地址开辟空间  local\_addr.sin\_family = AF\_INET; // 本地服务器IP有两个部分组成  local\_addr.sin\_addr.s\_addr = inet\_addr(local\_IP); // 本地服务器IP地址192.168.1.123  local\_addr.sin\_port = htons(UDP\_LOCAL\_PORT); // 本地服务器端口号9062  err = bind(sock\_fd, (struct sockaddr \*)&local\_addr, sizeof(local\_addr)); // 本地服务器地址绑定  if (err == -1)  {  closesocket(sock\_fd); // 绑定失败，关闭套接字  printk("failed to bind ip!\n");  return;  }  else  printk("succeed to bind ip!\n");    while (1)  {  struct sockaddr from;  socklen\_t fromlen;    memset(data\_buf, 0, UDP\_SERVER\_BUFSIZE); // 清除数据缓存区  err = recvfrom(sock\_fd, data\_buf, UDP\_SERVER\_BUFSIZE, 0, &from, &fromlen); // 接收数据  if (err > 0) // 正常接收并转发  {  printk("RECEIVE: %s \n", data\_buf);  sendto(sock\_fd, data\_buf, err, 0, &from, fromlen);  printk(" SEND: %s \n", data\_buf);  }  else if (err <= 0) // 接收错误继续接收  {  delay\_ms(100);  continue;  }  }  /\*  \* NEVER GO HERE!  \*/  closesocket(sock\_fd); // 关闭套接字  printk("udp\_server\_thread stop!\r\n"); // 结束  }  void udp\_server\_init(void) // 开启线程  {  sys\_thread\_new("udp\_server\_thread",  udp\_server\_thread,  NULL,  DEFAULT\_THREAD\_STACKSIZE,  DEFAULT\_THREAD\_PRIO + 1);  }  #endif |

（10）添加UDP客户端源文件”udp\_client.c”

|  |
| --- |
| #include "lwip\_test.h"  #if (TEST\_UDP\_CLIENT)  #include "lwip/sockets.h"  #include "lwip/err.h"  #include "lwip/sys.h"  static char data\_buf[UDP\_CLIENT\_BUFSIZE] = "this is a UDP test package";  static void udp\_client\_thread(void \*arg)  {  struct sockaddr\_in remote\_addr; // 定义远程服务器地址  int sock\_fd,err,count = 0; // 定义套接字，错误代码，计数  sock\_fd = socket(AF\_INET, SOCK\_DGRAM, 0); // 创建套接字（级别、类型、协议）  if (sock\_fd == -1)  {  printk("failed to create sock\_fd!\n");  return;  }  else  printk("succeed to create sock\_fd!\n");  memset(&remote\_addr, 0, sizeof(remote\_addr)); // 为远程服务器地址开辟空间  remote\_addr.sin\_family = AF\_INET; // 远程服务器IP有两个部分组成  remote\_addr.sin\_addr.s\_addr = inet\_addr(remote\_IP); // 远程服务器IP地址192.168.1.111  remote\_addr.sin\_port = htons(UDP\_REMOTE\_PORT); // 远程服务器端口号9063  while (1)  {  struct sockaddr from;  socklen\_t fromlen;  unsigned int ticks = get\_clock\_ticks(); // 获取时间值  memset(data\_buf, 0, UDP\_CLIENT\_BUFSIZE); // 清除数据缓存区  snprintf(data\_buf, UDP\_CLIENT\_BUFSIZE-1, "client ticks = %i.", ticks); // 【将字符串放在数据缓存区】  err = sendto(sock\_fd,data\_buf,strlen(data\_buf), // 向远程服务器发送数据  0,(struct sockaddr \*)&remote\_addr,  sizeof(remote\_addr));  if (err > 0)  printk("SEND: %s\r\n", data\_buf);  else if (err <= 0)  {  delay\_ms(1000);  continue;  }  delay\_ms(100);  memset(data\_buf, 0, UDP\_CLIENT\_BUFSIZE); // 清除数据缓存区  err = recvfrom(sock\_fd,data\_buf,UDP\_CLIENT\_BUFSIZE, // 接收数据  MSG\_WAITALL,&from,&fromlen);  // MSG\_PEEK阻塞第一条数据，MSG\_WAITALL阻塞每一条数据，MSG\_DONTWAIT非阻塞模式    if (err > 0)  printk(" RECEIVE: %s\r\n", data\_buf);  if (++count >= 5) /\* 结束标志 5次收发 \*/  break;  }    closesocket(sock\_fd); // 关闭套接字  printk("udp\_client\_thread stop!\r\n"); // 结束  }  void udp\_client\_init(void) // 开启线程  {  sys\_thread\_new("udp\_client\_thread",  udp\_client\_thread,  NULL,  DEFAULT\_THREAD\_STACKSIZE,  DEFAULT\_THREAD\_PRIO + 1);  }  #endif |

**(11)可选：修改“lwIP-1.4.1/port/ls1x\_ethernetif.c”文件中位于的ls1x\_initialize\_lwip（）函数，以实现观测ls1b开发板上GMAC的IP地址是否设置正确。**

①第509行添加：

|  |
| --- |
| printk("GMAC0\_IP is : %d.%d.%d.%d \n",ip0[0],ip0[1],ip0[2],ip0[3]); |
|  |

②第516行添加：

|  |
| --- |
| printk("GMAC0\_IP is default : 192.168.1.123 \n"); |
|  |

# 测试TCP与UDP协议

**开始测试之前，首先要进行下列准备操作：**

* 1. 用网线将PC机与ls1b开发板连接，建立局域网。
  2. 通过串口线将ls1b开发板UART5（控制台）与PC机相连并打开Putty准备调试。

③确保PC机器的IP与ls1b开发板的IP在同一网段下。本例程中ls1b开发板上GMAC0的IP地址被设置为192.168.1.123（如下图1所示），电脑以太网的IP地址为192.168.1.111（如下图2所示），由其子网掩码255.255.255.0可知PC机与ls1b开发板处于同一网段。



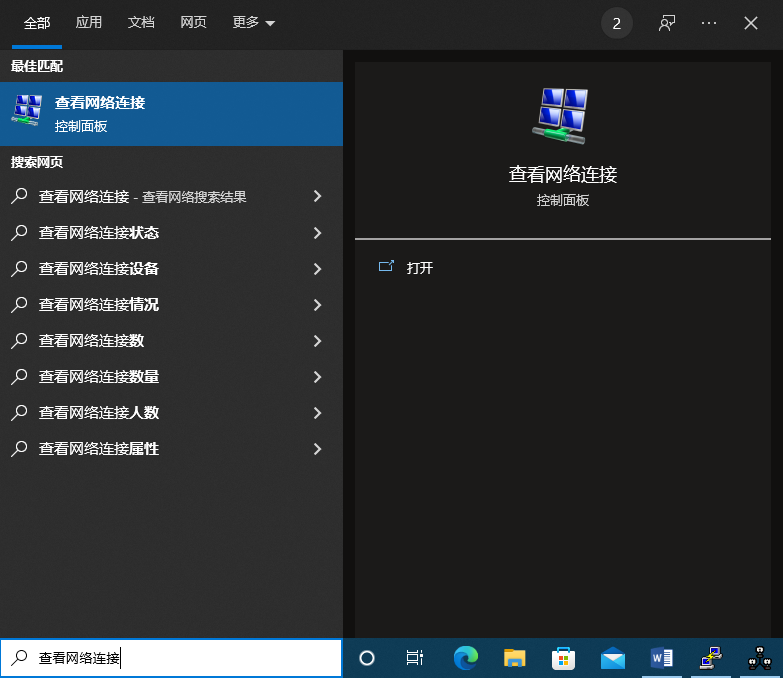
图1 主函数中定义的GMAC0地址



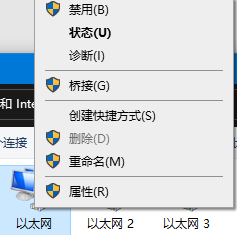
图2 PC机的IP地址

如何查看并更改PC机的IP:

首先按下“win+s”打开搜索栏，在搜索栏中输入“查看网络连接”，如下图所示。



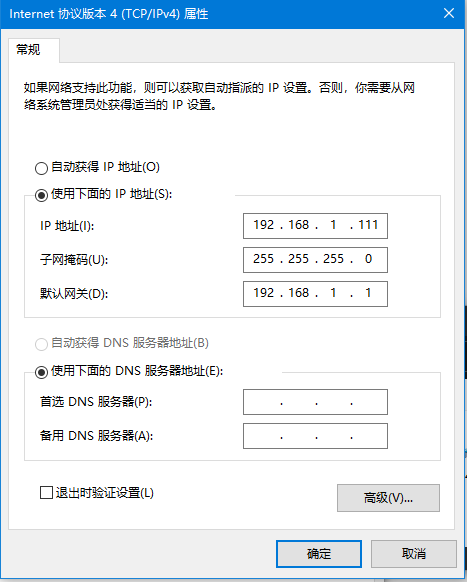
进入后，右键单击“以太网”，选择“属性”。



如下图所示，双击“internet协议版本4（TCP/IP）”。



最后即可配置IP地址。



③禁用其他以太网，以防造成影响。

如下图所示，在网络连接中右键单击其他以太网图标，选择“禁用”即可。



## 3.1 TCP服务器测试

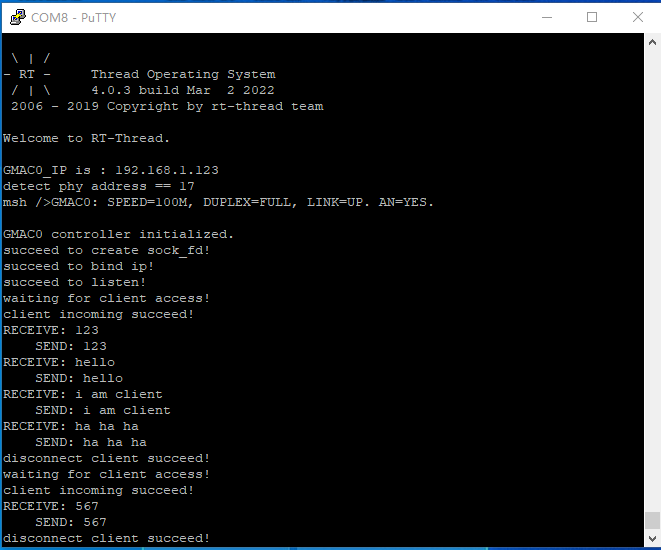
服务器（Ls-1B）：192.168.1.123：9060

客户端（PC机）：192.168.1.111

（1）修改“lwip\_test.h”文件中#define TEST\_TCP\_SERVER 为 1，其他三项为0。

（2）编译运行此工程。

（3）按下图所示配置网络调试助手，并接入服务器，发送数据。





## 3.2 TCP客户端测试

服务器（PC机）：192.168.1.111：9061

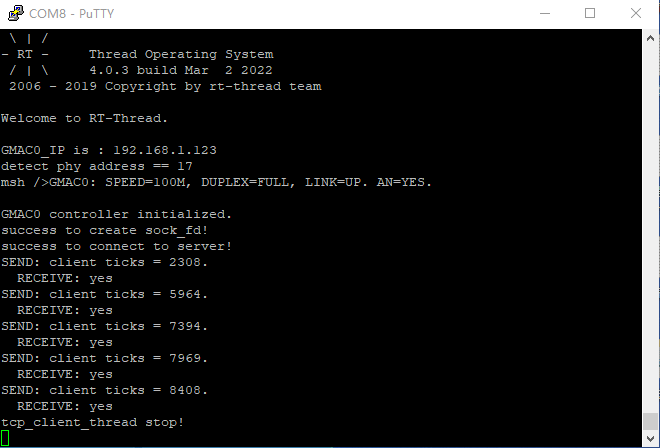
客户端（Ls-1B）：192.168.1.123

（1）修改“lwip\_test.h”文件中#define TEST\_TCP\_CLIENT 为 1，其他三项为0。

（2）按下图所示配置网络调试助手并打开，等待客户端连接。

（3）编译运行此工程，在服务器回复客户端信息。





## 3.3 UDP服务器测试

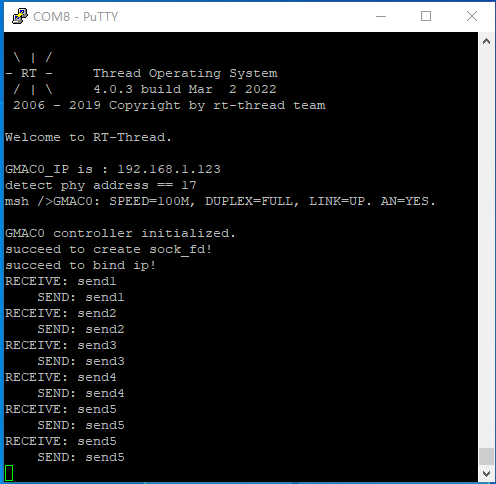
服务器（Ls-1B）：192.168.1.123：9062

客户端（PC机）：192.168.1.111：9062

（1）修改“lwip\_test.h”文件中#define TEST\_UDP\_SERVER 为 1，其他三项为0。

（2）编译运行此工程。

（3）按下图所示配置网络调试助手，并接入服务器，发送数据。





## 3.4 UDP客户端测试

服务器（PC机）：192.168.1.111：9063

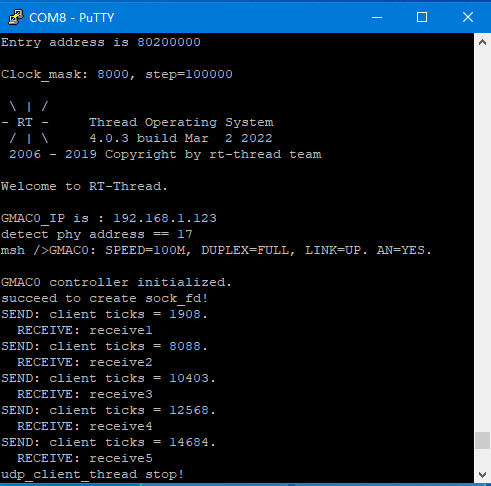
客户端（Ls-1B）：192.168.1.123

（1）修改“lwip\_test.h”文件中#define TEST\_UDP\_CLIENT 为 1，其他三项为0。

（2）按下图所示配置网络调试助手并打开，等待客户端连接。

（3）编译运行此工程，在服务器回复客户端信息。





本质上UDP是不分服务器与客户端的，但是双向的信息传递，总要有一方发起。所以主动发送信息的一方即为客户端。

# 4. 两个ls1b开发板通信

## 4.1 创建另一个工程

正如上一节所示，我们分别实现了TCP协议的服务器、客户端与UDP协议的服务器、客户端。那么如何让两块开发板进行通信呢？

让其中一块开发板代替上一节测试中PC机的角色即可。当然，整个工程是相同的只是IP地址和端口不同。解决办法如下：首先我们将第二节创建的工程” RT\_lwip\_TCP\_UDP”拷贝一份，将其重命名为“RT\_lwip\_TCP\_UDP\_PC”，然后再更改其IP与端口，具体操作如下：

（1）更改当前开发板的IP地址GMAC0：

将“main.c”文件下第22行“static unsigned char gmac0\_ip[4] = {192, 168, 1, 123};”更改为“static unsigned char gmac0\_ip[4] = {192, 168, 1, 111};”这样就将“RT\_lwip\_TCP\_UDP\_PC”工程中下GMAC0的IP地址更改为了“192, 168, 1, 111”。结果如下图所示：



（2）更改本地IP与远程IP，保证可以交叉连接：

打开“lwip\_test.c”文件，将本地IP“192.168.1.123”与远程IP“192.168.1.111”互换，结果如下图所示：



（3）更改端口号，保证可以交叉连接：

更改TCP服务器和客户端的端口号：将“lwip\_test.h”文件下定义的TCP本地端口与TCP远程端口互换，结果如下图所示：



更改UDP服务器和客户端的端口号：将“lwip\_test.h”文件下定义的TCP本地端口与TCP远程端口互换，结果如下图所示：



至此，该工程搭建完毕。

## 4.2 两块开发板通信

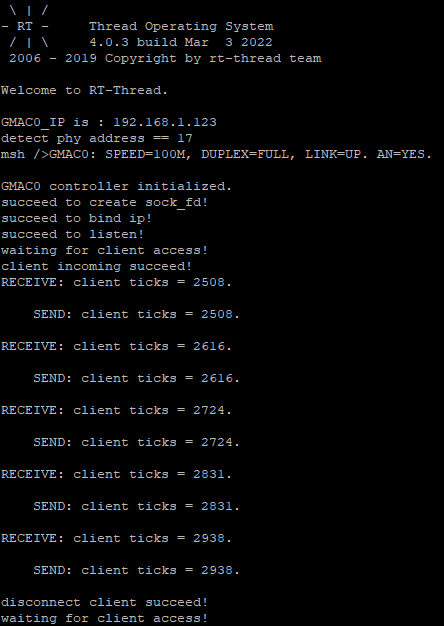
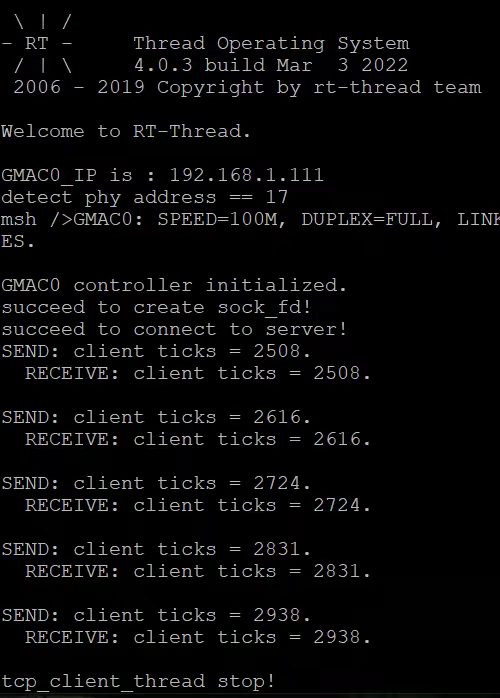
### 4.2.1 TCP通信

（1）准备工作：将两块开发板的EJTAG线分别连接到两台PC机（PC机1和PC机2），两块开发板上的UART5（控制台）通过串口线分别接入两台电脑，并根据电脑虚拟出的端口号打开“PuTTY”软件观测其控制台。连接电源线并给开发板上电。

（2）操作PC机1：打开工程“RT\_lwip\_TCP\_UDP”，在该工程下的“lwip\_test.h”文件中，通过修改宏定义，将开发板设为TCP协议的服务器（具体操作第2节中有讲），然后编译运行。

（3）操作PC机2：打开工程“RT\_lwip\_TCP\_UDP\_PC”，在该工程下的“lwip\_test.h”文件中，通过修改宏定义，将开发板设为TCP协议的客户端（具体操作第2节中有讲），然后编译运行。

（4）此时可通过控制台观测到服务器与客户端之间的通信，如下图所示（左侧为TCP服务器，右侧为TCP客户端）。

（5）总结如下：

|  |  |  |
| --- | --- | --- |
| 属性 | PC机1 | PC机2 |
| 打开工程 | RT\_lwip\_TCP\_UDP | RT\_lwip\_TCP\_UDP\_PC |
| 角色 | TCP服务器 | TCP客户端 |
| GMAC0 | 192.168.1.123 | 192.168.1.111 |
| 端口 | 9060 | 随机分配 |
| 动作 | 绑定本地服务器端口（本地IP） | 向远程服务器申请连接（服务器IP） |

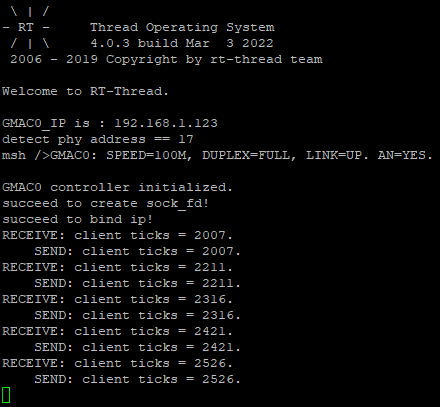
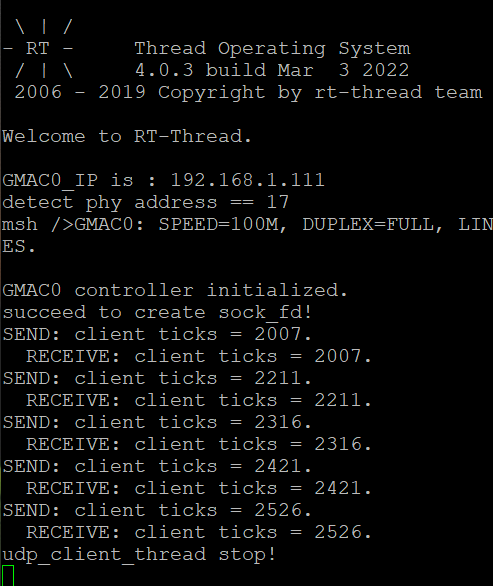
### 4.2.2 UDP通信

（1）准备工作：将两块开发板的EJTAG线分别连接到两台PC机（PC机1和PC机2），两块开发板上的UART5（控制台）通过串口线分别接入两台电脑，并根据电脑虚拟出的端口号打开“PuTTY”软件观测其控制台。连接电源线并给开发板上电。

（2）操作PC机1：打开工程“RT\_lwip\_TCP\_UDP”，在该工程下的“lwip\_test.h”文件中，通过修改宏定义，将开发板设为UDP协议的服务器（具体操作第2节中有讲），然后编译运行。

（3）操作PC机2：打开工程“RT\_lwip\_TCP\_UDP\_PC”，在该工程下的“lwip\_test.h”文件中，通过修改宏定义，将开发板设为UDP协议的客户端（具体操作第2节中有讲），然后编译运行。

（4）此时可通过控制台观测到服务器与客户端之间的通信，如下图所示（左侧为UDP服务器，右侧为UDP客户端）。

（5）总结如下：

|  |  |  |
| --- | --- | --- |
| 属性 | PC机1 | PC机2 |
| 打开工程 | RT\_lwip\_TCP\_UDP | RT\_lwip\_TCP\_UDP\_PC |
| 角色 | UDP服务器 | UDP客户端 |
| GMAC0 | 192.168.1.123 | 192.168.1.111 |
| 端口 | 9062 | 随机分配 |
| 动作 | 绑定本地服务器端口（本地IP） | 向远程服务器申请连接（服务器IP） |

### 4.2.3 PC机2作为服务器

|  |  |  |
| --- | --- | --- |
| 属性 | PC机1 | PC机2 |
| 打开工程 | RT\_lwip\_TCP\_UDP | RT\_lwip\_TCP\_UDP\_PC |
| 角色 | TCP客户端 | TCP服务器 |
| GMAC0 | 192.168.1.123 | 192.168.1.111 |
| 端口 | 随机分配 | 9061 |
| 动作 | 向远程服务器申请连接（服务器IP） | 绑定本地服务器端口（本地IP） |

|  |  |  |
| --- | --- | --- |
| 属性 | PC机1 | PC机2 |
| 打开工程 | RT\_lwip\_TCP\_UDP | RT\_lwip\_TCP\_UDP\_PC |
| 角色 | UDP客户端 | UDP服务器 |
| GMAC0 | 192.168.1.123 | 192.168.1.111 |
| 端口 | 随机分配 | 9063 |
| 动作 | 向远程服务器申请连接（服务器IP） | 绑定本地服务器端口（本地IP） |

【注】：先运行服务器再运行客户端。