Technical Note

LwIP 的 RAW API 接口及编程指南

TCP/IP 协议栈应用

V0.01 Date:2008/10/27

工程技术笔记

类别	内容
关键词	LwIP，RAW API
摘　要	本文详细讲述了 LwIP 在无操作系统支持环境下的 API 函数介绍及编程应用。首先，介绍了 RAW API 的特点及优缺点，然后逐个介绍了 LwIP 提供的所有的 RAW API 函数，最后通过实例的形式介绍了这些 API 函数具体编程的方法。

广州致远电子有限公司

ZHIYUAN ELECTRONICS

修订历史

版本	日期	原因
V0.00	2008/10/10	创建文档
V0.01	2008/10/27	修改了 UDP 服务器的程序。因为作为服务器的主机并不一定要预先知道客户机的 IP 地址及端口号，本版本程序从回调函数的参数中获取了远程客户机的 IP 地址及端口号等信息。

销售与服务网络（一）

广州周立功单片机发展有限公司

地址：广州市天河北路 689 号光大银行大厦 12 楼 F4 邮编：510630
电话：(020)38730916 38730917 38730972 38730976 38730977
传真：(020)38730925
网址：www.zlgmcu.com

广州专卖店

地址：广州市天河区新赛格电子城 203-204 室
电话：(020)87578634 87569917
传真：(020)87578842

南京周立功

地址：南京市珠江路 280 号珠江大厦 2006 室
电话：(025)83613221 83613271 83603500
传真：(025)83613271

北京周立功

地址：北京市海淀区知春路 113 号银网中心 A 座 1207-1208 室（中发电子市场斜对面）
电话：(010)62536178 62536179 82628073
传真：(010)82614433

重庆周立功

地址：重庆市石桥铺科园一路二号大西洋国际大厦（赛格电子市场）1611 室
电话：(023)68796438 68796439
传真：(023)68796439

杭州周立功

地址：杭州市天目山路 217 号杭州电子科技大楼 502 室
电话：(0571) 28139611 28139612 28139613
传真：(0571) 28139621

成都周立功

地址：成都市一环路南二段 1 号数码同人港 401 室（磨子桥立交西北角）
电话：(028)85439836 85437446
传真：(028)85437896

深圳周立功

地址：深圳市深南中路 2070 号电子科技大厦 C 座 4 楼 D 室
电话：(0755)83781788（5 线）
传真：(0755)83793285

武汉周立功

地址：武汉市洪山区广埠屯珞瑜路 158 号 12128 室（华中电脑数码市场）
电话：(027)87168497 87168297 87168397
传真：(027)87163755

上海周立功

地址：上海市北京东路 668 号科技京城东座 7E 室
电话：(021)53083452 53083453 53083496
传真：(021)53083491

西安办事处

地址：西安市长安北路 54 号太平洋大厦 1201 室
电话：(029)87881296 83063000 87881295
传真：(029)87880865

销售与服务网络（二）

广州致远电子有限公司

地址：广州市天河区车陂路黄洲工业区 3 栋 2 楼
邮编：510660
传真：(020)38601859
网址：www.embedtools.com （嵌入式系统事业部）
www.embedcontrol.com （工控网络事业部）
www.ecardsys.com （楼宇自动化事业部）

技术支持：

CAN-bus：

电话：(020)22644381 22644382 22644253
邮箱：can.support@embedcontrol.com

iCAN 及模块：

电话：(020)28872344 22644373
邮箱：ican@embedcontrol.com

MiniARM：

电话：(020)28872684 28267813
邮箱：miniarm.support@embedtools.com

以太网及无线：

电话：(020)22644380 22644385 22644386
邮箱：wireless@embedcontrol.com
ethernet.support@embedcontrol.com

编程器：

电话：(020)22644371
邮箱：programmer@embedtools.com

分析仪器：

电话：(020)22644375 28872624 28872345
邮箱：tools@embedtools.com

ARM 嵌入式系统：

电话：(020)28872347 28872377 22644383 22644384
邮箱：arm.support@zlgmcu.com

楼宇自动化：

电话：(020)22644376 22644389 28267806
邮箱：mjs.support@ecardsys.com
mifare.support@zlgmcu.com

销售：

电话：(020)22644249 22644399 22644372 22644261 28872524
28872342 28872349 28872569 28872573 38601786

维修：

电话：(020)22644245

目 录

1. 简介

LwIP 为我们提供了两种应用程序接口（API 函数）来实现 TCP/IP 协议栈，它们分别是：

- low-level "core" / "callback" or "raw" API.

——低水平的、基于回调函数的 API（后面直接称 RAW API）

- higher-level "sequential" API.

——高水平的、连续的 API（后面直接称 sequential API）

在使用 LwIP 栈编程的时候，sequential API 为我们提供了一种通用的方法，它与 BSD 标准的 socket API 非常相似,程序的执行过程同样是基于"open-read-write-close"模型的。从本质上讲，TCP/IP 协议栈的通信过程是事件驱动的，因此，TCP/IP 的代码和用户应用程序的代码，应该在不同的线程里面。

下面，这篇文档的剩余部分我们来讨论 RAW API，关于 sequential API 的编程使用方法作者也有一个专门的说明文档，读者可以查阅相关的文档，这里不做介绍。

使用 RAW API 进行 TCP/IP 编程，可以使应用程序的代码和 TCP/IP 协议栈的代码很好地结合起来。程序的执行机制是以回调函数为基础的事件驱动的，同时回调函数也是被 TCP/IP 代码直接调用的。TCP/IP 代码和应用程序的代码运行在同一个线程里面。sequential API 接口函数则是一种高起点的函数，它不适合于应用在比较小的嵌入式系统中，因为它的实现要求应用程序必须支持多线程。

RAW API 接口函数不仅在程序代码的执行时间上更快，而且在运行中它也占用更少的内存资源。唯一的缺点是应用程序的编写比较困难，并且代码较难理解。尽管如此，在 CODE 和 RAM 都较小的嵌入式系统中，这也是我们优先考虑采用的方法。

当然，在不同的应用程序中，这两种 API 我们可以同时采用。实际上，sequential API 就是一种利用 RAW API 来实现的一种属于协议本体的应用程序。

2. RAW API 参考手册

2.1 简介——回调函数

程序的执行是靠回调函数来驱动的。每一个回调函数也只不过是一个能够直接被 TCP/IP 代码调用的普通的 C 语言函数。每一个回调函数的调用都是传递一个当前连接 UDP 或 TCP 的状态。另外，为了使应用程序有一个明确的执行状态，回调函数的指定是可编程的，并且是独立于 TCP/IP 状态之外的。

剩下的部分我们将逐个来介绍 LwIP 提供的 RAW API 函数。

2.2 应用程序状态设置函数

1. tcp_arg()

该函数用于传递给应用程序的具体状态，在控制块标志建立以后调用，即在函数 tcp_new()(请见表 2.2)调用之后才能调用，该函数的详细描述请见表 2.1。

表 2.1 函数 tcp_arg()

功能	指定应该传递给所有回调函数的应用程序的具体状态
原型	void tcp_arg(struct tcp_pcb *pcb, void *arg)
参数	pcb： 当前 TCP 连接的控制块 arg： 需要传递给回调函数的参数
返回	无

2.3 建立 TCP 连接函数

建立连接的函数同 sequential API 以及 BSD 标准的 socket API 都非常相似。一个新的 TCP 连接的标志(实质上是一个协议控制块-PCB)由 tcp_new()函数来创建。连接创建后，该 PCB 可以进入监听状态，等待数据接收的连接信号，也可以直接连接另外一个主机。

1. tcp_new()

该函数在定义一个tcp_pcb控制块后应该首先被调用，以建立该控制块的连接标志。该函数的详细描述请见表 2.2。

表 2.2 函数 tcp_new()

功能	建立一个新的连接标志(pcb)
原型	struct tcp_pcb *tcp_new(void)
参数	无
返回	pcb: 正常建立了连接标志，返回建立的 pcb NULL：新的 pcb 内存不可用时

2. tcp_bind()

该函数用户绑定本地的IP地址和端口号，用户可以将其绑定在一个任意的本地IP地址上，它也只能在函数tcp_new()(请见表 2.2)调用之后才能调用。该函数的详细描述请见表 2.3。

表 2.3 函数 tcp_bind()

功能	绑定本地 IP 地址和端口号
原型	err_t tcp_bind (struct tcp_pcb *pcb, struct ip_addr *ipaddr, u16_t port)
参数	pcb： 准备绑定的连接，类似于 BSD 标准中的 Sockets ipaddr：绑定的 IP 地址。如果为 IP_ADDR_ANY，则将连接绑定到所有的本地 IP 地址上 port： 绑定的本地端口号。注意：千万不要和其它的应用程序产生冲突
返回	ERR_OK： 正确地绑定了指定的连接 ERR_USE： 指定的端口号已经绑定了一个连接，产生了冲突

3. tcp_listen()

当一个正在请求的连接被接收时，由tcp_accept()(请见表 2.7)函数指定的回调函数将会被调用。当然，在调用本函数前，必须首先调用函数tcp_bind()(请见表 2.3)来绑定一个本地的IP地址和端口号。该函数的详细描述请见表 2.4。

表 2.4 函数 tcp_listen()

功能	使指定的连接开始进入监听状态
原型	struct tcp_pcb *tcp_listen (struct tcp_pcb *pcb)
参数	pcb：指定将要进入监听状态的连接
返回	pcb: 返回一个新的连接标志 pcb，它作为一个参数传递给将要被分派的函数。这样做的原因是处于监听状态的连接一般只需要较小的内存，于是函数 tcp_listen()就会收回原始连接的内存，而重新分配一个较小内存块供处于监听状态的连接使用。 NULL： 监听状态的连接的内存块不可用时，返回 NULL。如果这样的话，作为参数传递给函数 tcp_listen()的 pcb 所占用的内存将不能够被分配。

4. tcp_listen_with_backlog()

该函数同tcp_listen()一样，但是该函数将限制在监听队列中未处理的连接的数量，这是通过参数backlog来实现的。要使用该函数，需要在配置文件lwipopts.h中设置TCP_LISTEN_BACKLOG=1。该函数的详细描述请见表 2.5。

表 2.5 函数 tcp_listen_with_backlog()

功能	使指定的连接开始进入监听状态，但将会限制监听队列中连接的数量
原型	struct tcp_pcb *tcp_listen_with_backlog(struct tcp_pcb *pcb, u8_t backlog)
参数	pcb: 指定将要进入监听状态的连接 backlog: 限制监听队列中连接的数量
返回	pcb: 返回一个新的连接标志 pcb，它作为一个参数传递给将要被分派的函数。这样做的原因是处于监听状态的连接一般只需要较小的内存，于是函数 tcp_listen()就会收回原始连接的内存，而重新分配一个较小内存块供处于监听状态的连接使用。 NULL： 监听状态的连接的内存块不可用时，返回 NULL。如果这样的话，作为参数传递给函数 tcp_listen()的 pcb 所占用的内存将不能够被分配。

5. tcp_accepted()

这个函数通常在“accept”的回调函数中被调用。它允许LwIP去执行一些内务工作，例

如，将新来的连接放入到监听队列中，以等待处理。该函数的详细描述请见表 2.6。

表 2.6 函数 tcp_accepted()

功能	通知 LwIP 一个新来的连接已经被接收
原型	void tcp_accepted(struct tcp_pcb *pcb)
参数	pcb: 已经被接收的连接
返回	无

6. tcp_accept()

当处于监听的连接与一个新来的连接连接上后， 该函数指定的回调函数将被调用。通常在tcp_listen()(请见表 2.4)函数调用之后调用。该函数的详细描述请见表 2.7。

表 2.7 函数 tcp_accept()

功能	指定处于监听状态的连接接通后将要调用的回调函数
原型	void tcp_accept(struct tcp_pcb *pcb, err_t (* accept)(void *arg, struct tcp_pcb *newpcb, err_t err))
参数	pcb： 指定一个处于监听状态的连接 accept：指定连接接通后将要调用的回调函数
返回	无

7. tcp_connect()

请求参数pcb指定的连接连接到远程主机，并发送打开连接的最初的SYN段。函数tcp_connect()调用后立即返回，它并不会等待连接一定要正确建立。如果当连接正确建立，那么它会直接调用第四个参数指定的函数(connected参数)。相反地，如果连接不能够被正确建立，这原因可能是远程主机拒绝连接，也可能是远程主机不应答，无论是什么原因，都会调用connected函数来设置相应的参数err。该函数的详细描述请见表 2.8。

表 2.8 函数 tcp_connect()

功能	请求指定的连接连接到远程主机，并发送打开连接的最初的 SYN 段
原型	err_t tcp_connect(struct tcp_pcb *pcb, struct ip_addr *ipaddr, u16_t port, err_t (* connected)(void *arg, struct tcp_pcb *tpcb, err_t err))
参数	pcb： 指定一个连接(pcb) ipaddr： 指定连接远程主机的 IP 地址 port： 指定连接远程主机的端口号 connected：指定连接正确建立后调用的回调函数
返回	ERR_MEM：当访问 SYN 段的内存不可用时，即连接没有成功建立 ERR_OK： 当 SYN 被正确地访问时，即连接成功建立

2.4 TCP 数据发送函数

LwIP 通过调用函数 tcp_write()来发送 TCP 数据。当数据被成功地发送到远程主机后，应用程序将会收到应答从而去调用一个指定的回调函数。

1. tcp_write()

该函数功能是发送TCP数据，但是并不是一经调用，就立即发送数据，而是将指定的数据放入到发送队列，由协议内核来决定发送。发送队列中可用字节的大小可以通过函数tcp_sndbuf()来重新获得。使用这个函数的一个比较恰当的方法是以函数tcp_sndbuf()返回的字节大小来发送数据。如果函数返回ERR_MEM，则应用程序就等待一会，直到当前发送队列中的数据被远程主机成功地接收，然后在尝试发送下一个数据。该函数的详细描述请见表2.9。

表 2.9 函数 tcp_write()

功能	发送 TCP 数据
原型	err_t tcp_write(struct tcp_pcb *pcb, void *dataptr, u16_t len, u8_t copy)
参数	pcb： 指定所要发送的连接(pcb) dataptr：是一个指针，它指向准备发送的数据 len： 指定要发送数据的长度 copy： 这是一个逻辑变量，它为 0 或者 1，它指定是否分配新的内存空间，而把要发送的数据复制进去。如果该参数为 0，则不会为发送的数据分配新的内存空间，因而对发送数据的访问只能通过指定的指针
返回	ERR_MEM：如果数据的长度超过了当前发送数据缓冲区的大小或者将要发送的段队列的长度超过了文件 lwipopts.h 中定义的上限(即最大值)，则函数 tcp_write()调用失败，返回 ERR_MEM ERR_OK：数据被正确地放入到发送队列中，返回 ERR_OK

2. tcp_sent ()

该函数用于设定远程主机成功接收到数据后调用的回调函数，通常也在函数tcp_listen()(请见表 2.4)之后调用。该函数的详细描述请见表 2.10。

表 2.10 函数 tcp_sent()

功能	指定当远程主机成功地接收到数据后，应用程序调用的回调函数
原型	void tcp_sent(struct tcp_pcb *pcb, err_t (* sent)(void *arg, struct tcp_pcb *tpcb, u16_t len))
参数	pcb： 指定一个与远程主机相连接的连接(pcb) sent： 指定远程主机成功地接收到数据后调用的回调函数。“len”作为参数传递给回调函数，给出上一次已经被确认的发送的最大字节数。
返回	无

2.5 TCP 数据接收函数

TCP 数据的接收是基于回调函数的，当新的数据到达时，应用程序指定的回调函数将会被调用。当应用程序接收到数据后，它必须立即调用函数 tcp_recved()来指示接收数据的大小。

1. tcp_recv ()

该函数用于指定当有新的数据接收到时调用的回调函数，通常在函数tcp_accept()(表 2.7)指定的回调函数中调用。该函数的详细描述请见表 2.11。

表 2.11 函数 tcp_recv()

功能	指定当新的数据接收到时调用的回调函数
原型	void tcp_recv (struct tcp_pcb *pcb, err_t (* recv)(void *arg, struct tcp_pcb *tpcb, struct pbuf *p, err_t err))
参数	pcb： 指定一个与远程主机相连接的连接(pcb) recv： 指定当新的数据接收到时调用的回调函数。该回调函数可以通过传递一个NULL的pbuf结构用来指示远程主机已经关闭连接。如果没有错误发生，则回调函数返回ERR_OK，并且必须释放掉pbuf结构。否则，如果函数的调用中发生错误，那么千万不要释放该结构，以便LwIP内核可以保存该结构，从而等待以后处理。
返回	无

2. tcp_recved ()

当应用程序接收到数据的时候该函数必须被调用，用于获取接收到的数据的长度，即该函数应该在函数tcp_recv()(表 2.11)指定的回调函数中调用。该函数的详细描述请见表 2.12。

表 2.12 函数 tcp_recved()

功能	获取接收到的数据的长度
原型	void tcp_recved(struct tcp_pcb *pcb, u16_t len)
参数	pcb： 指定一个与远程主机相连接的连接(pcb) len： 获取接收到的数据的长度
返回	无

2.6 应用程序轮询工作原理及相关函数

当连接是空闲的时候(也就是说没有数据发送与接收)，LwIP 将会通过一个回调函数来重复地轮询应用程序。这可以用作一个看门狗定时器来删除连接当连接保持空闲的时间太长的时候，也可以用作一种等待内存变为可用状态的方法。例如，如果因为内存不可用而导致调用函数 tcp_write()失败了，当连接空闲一段时间以后，应用程序就会利用轮询功能来重新调用函数 tcp_write()。

1. tcp_poll()

当使用LwIP的轮询功能时必须调用该函数，用于指定轮询的时间间隔及轮询时应该调用的回调函数。该函数的详细描述请见表 2.13。

表 2.13 函数 tcp_poll()

功能	指定轮询的时间间隔以及轮询应用程序时应该调用的回调函数
原型	void tcp_poll(struct tcp_pcb *pcb, err_t (* poll)(void *arg, struct tcp_pcb *tpcb), u8_t interval)
参数	pcb： 指定一个连接(pcb) poll： 指定轮询应用程序时应该调用的回调函数 interval：指定轮询的时间间隔。时间间隔应该以 TCP 的细粒度定时器为单位，典型的设置是每秒钟两次。把参数“interval”设置为 10 意味着应用程序将每 5 秒钟轮询一次。
返回	无

2.7 关闭与中止连接的函数

1. tcp_close

表 2.14 函数 tcp_close()

功能	关闭一个指定的 TCP 连接，调用该函数后，TCP 代码将会释放(删除)pcb 结构
原型	err_t tcp_close(struct tcp_pcb *pcb)
参数	pcb： 指定一个需要关闭的连接(pcb)
返回	ERR_MEM：当需要关闭的连接没有可用的内存时，该函数返回 ERR_MEM。如果这样的话，应用程序将通过事先确立的回调函数或者是轮询功能来等待及重新关闭连接 ERR_OK：连接正常关闭。

2. tcp_abort()

该函数通过向远程主机发送一个RST(复位)段来中止连接。pcb结构将会被释放。该函数是不会失败的，它一定能完成中止的目的。该函数的详细描述请见表 2.15。

如果连接是因为一个错误而产生了中止，则应用程序会通过回调函数灵敏地处理这个事件。通常发送错误而引起的连接中止都是因为内存资源短缺引起的。设置处理错误的回调函数是通过函数tcp_err()(请见表 2.16)来完成。

表 2.15 函数 tcp_abort()

功能	中止一个指定的连接(pcb)
原型	void tcp_abort(struct tcp_pcb *pcb)
参数	pcb： 指定一个需要关闭的连接(pcb)
返回	无

3. tcp_err()

该函数用于指定处理错误的回调函数。一个可靠的优秀的应用程序一般都要处理可能出现的错误，如内存不可用等，这就需要调用该函数来指定一个回调函数来获取错误信息。该函数的详细描述请见表 2.16。

表 2.16 函数 tcp_err()

功能	指定处理错误的回调函数
原型	void tcp_err(struct tcp_pcb *pcb, void (* err)(void *arg, err_t err))
参数	pcb： 指定需要处理的发送错误的连接(pcb) err： 指定发送错误时调用的回调函数。因为 pcb 结构可能已经被删除了，所以在处理错误的回调函数中 pcb 参数不可能传递进来。
返回	无

2.8 底层 TCP 接口

TCP 在系统的底层为我们提供了一个简单的接口。在系统初始化的过程中，函数 tcp_init()必须被在调用其他的 TCP 函数之前被首先调用。另外，在系统正常运行的过程中，两个定时器函数 tcp_fasttmr()和 tcp_slowtmr()必须以固定的时间间隔有规律地被调用。函数 tcp_fasttmr()应该每 TCP_FAST_INTERVAL 毫秒被调用一次,而函数 tcp_slowtmr()应该每 TCP_SLOW_INTERVAL 毫秒被调用一次(常量 TCP_FAST_INTERVAL 和 TCP_SLOW_INTERVAL 都在文件 tcp.h 中定义)。

上面这几个函数的调用都在系统初始化的过程中完成，并且需要用到一个硬件定时器，用来处理周期性的事件。

2.9 UDP 接口函数

UDP 接口与 TCP 接口比较相似，但其复杂性相对较低，这就意味着 UDP 接口将会非常地简单。

1. udp_new()

该函数用于建立一个用于UDP通信的UDP控制块（pcb)，但是这个pcb并没有被激活，除非该pcb已经被绑定到一个本地地址上或者连接到一个固定地址的远程主机。在定义一个udp_pcb控制块后该函数应该首先被调用，以建立该控制块的连接标志。该函数的详细描述请见表 2.17。

表 2.17 函数 udp_new()

功能	建立一个用于 UDP 通信的 UDP 控制块（pcb)
原型	struct udp_pcb *udp_new(void)
参数	无
返回	udp_pcb：建立的 UDP 连接的控制块(pcb)

2. udp_remove()

该函数用于删除一个指定的连接，通常是控制块在建立成功后，即在函数udp_new()(请见表 2.17)调用之后，当不需要该网络连接来通信了，就需要将其删除，以释放该连接(pcb)所占用的资源。该函数的详细描述请见表 2.18。

表 2.18 函数 udp_remove()

功能	删除并释放掉一个 udp_pcb
原型	void udp_remove(struct udp_pcb *pcb)
参数	pcb：指定要删除的连接(pcb)
返回	无

3. udp_bind()

该函数用户绑定本地的IP地址和端口号，用户可以将其绑定在一个任意的本地IP地址上，它也只能在函数udp_new()(请见表 2.17)调用之后才能调用。该函数的详细描述请见表 2.19。

表 2.19 函数 udp_bind()

功能	为指定的连接绑定本地 IP 地址和端口号
原型	err_t udp_bind(struct udp_pcb *pcb, struct ip_addr *ipaddr, u16_t port)
参数	pcb： 指定一个连接(pcb) ipaddr：绑定的本地 IP 地址。如果为 IP_ADDR_ANY，则将连接绑定到所有的本地 IP 地址上 port： 绑定的本地端口号。注意：千万不要和其它的应用程序产生冲突
返回	ERR_OK： 正确地绑定了指定的连接 ERR_USE： 指定的端口号已经绑定了一个连接，产生了冲突

4. udp_connect()

该函数将一个指定的连接(pcb)连接到远程主机。由于UDP通信是面向无连接的，所以这不会参数任何的网络流量(网络数据收发)，它仅仅是设置了一个远程连接的IP地址和端口号。该函数的详细描述请见表 2.20。

表 2.20 函数 udp_connect()

功能	将参数“pcb”指定的连接控制块连接到远程主机
原型	err_t udp_connect(struct udp_pcb *pcb, struct ip_addr *ipaddr, u16_t port)
参数	pcb： 指定一个连接(pcb) ipaddr：设置连接的远程主机 IP 地址 port： 设置连接的远程主机端口号
返回	ERR_OK：正确连接到远程主机 其它值： LwIP 的一些错误代码标志，表示连接没有正确建立

5. udp_disconnect()

该函数关闭参数“pcb”指定的连接，同函数udp_connect()作用相反。由于UDP通信是面向无连接的，所以这个函数同样不会参数任何的网络流量（(网络数据收发)，它仅仅是删除了远程连接的地址。该函数的详细描述请见表 2.21。

表 2.21 函数 udp_disconnect()

功能	关闭参数“pcb”指定的连接，同函数 udp_connect()作用相反
原型	void udp_disconnect(struct udp_pcb *pcb)
参数	pcb：指定要删除的连接(pcb)
返回	无

6. udp_send()

该函数使用UDP协议发送pbuf p指向的数据。在需要发送数据时调用，发送后，该pbuf结构并没有被释放。调用该函数后，数据包将被发送到存放在pcb中的当前指定的IP地址和端口号上。如果该pcb没有连接到一个固定的端口号，那么该函数将会自动随机地分配一个端口号，并将数据包发送出去。通常，在调用前都会先调用函数udp_connect()(请见表 2.20)。该函数的详细描述请见表 2.22。

表 2.22 函数 udp_send()

功能	使用 UDP 协议发送 pbuf p 指向的数据
原型	err_t udp_send(struct udp_pcb *pcb, struct pbuf *p)
参数	pcb：指定发送数据的连接(pcb) p： 包含需要发送数据的 pbuf 链
返回	ERR_OK：数据包成功发送，没有任何错误发生 ERR_MEM：内存不可用 ERR_RTE：不能找到到达远程主机的路由 其它值：其它的一些错误码，都表示发送了错误

7. udp_sendto()

该函数同udp_send()作用一样，但是它指定了发送的目的主机IP地址和端口号，相当于udp_connect()和函数udp_send()合在一起使用的效果。但是，如果在调用该函数前已经调用过函数udp_connect()，那么发送目的主机的IP地址和端口号将以本函数指定的为准，由函数udp_connect()指定的将会被刷新。该函数的详细描述请见表 2.23。

表 2.23 函数 udp_sendto()

功能	向具有指定的 IP 地址和端口号远程主机发送 UDP 数据
原型	err_t udp_sendto(struct udp_pcb *pcb, struct pbuf *p, struct ip_addr *dst_ip, u16_t dst_port)
参数	pcb： 指定发送数据的连接(pcb) p： 包含需要发送数据的 pbuf 链 dst_ip：发送数据的远程主机 IP 地址 dst_port：发送数据的远程主机端口号
返回	同函数 udp_send()的返回值一样

8. udp_recv()

该函数用于指定当有新的UDP数据接收到时被调用的回调函数，回调函数将的参数将传递进远程主机的IP地址、端口号及接收到的数据等信息。该函数的详细描述请见表 2.24。

表 2.24 函数 udp_recv()

功能	指定一个接收到 UDP 数据包时被调用的回调函数
原型	void udp_recv(struct udp_pcb *pcb, void (* recv)(void *arg, struct udp_pcb *upcb, struct pbuf *p, struct ip_addr *addr, u16_t port), void *recv_arg)
参数	pcb： 指定一个连接(pcb) recv： 指定数据包接收到时的回调函数 recv_arg：传递给回调函数的参数
返回	无

3. 应用程序实例

上一章我们介绍了 LwIP 的一些比较底层的 RAW API 函数，这一章我们将使用这些函数来进行简单的编程，通过几个实例来介绍 LwIP 的应用。需要事先声明的是，在实验前，应该先修改在 lwip-1.3.0\ports 目录下文件 Eth_Config.h 中的设置，修改一套合适的以太网的 MAC 地址、IP 地址、网关地址及子网掩码。本章的实验设定的结果如下：

```
#define My_Mac_ID     {0X00,0x14,0x97,0x0F,0x1D,0xE3} //   存储以太网控制器的物理地址,即 MAC 地址
#define IP_MARK_ID    {255,255,255,0}                 //   255.255.255.0,子网掩码
#define MY_IP_ID      {192,168,1,25}                  //   以太网通信的 IP 地址
#define MY_GATEWAY_ID     {192,168,1,254}             //   以太网通信的网关地址
```

下面，在来介绍这几个编程的实例。

注意：本章的所有程序都是在 Luminary 公司生产的群星系列单片机，在 EK-LM3S6965 评估板上调试通过的，在广州致远电子有限公司生产的 EasyARM 8962 开发板上不需要做任何的修改，也可以正常运行。但通信的上位机和目标板的网关地址和子网掩码必须相同，IP 地址一定不能冲突。

3.1 UDP 服务器通信实例

UDP 在网络体系结构中属于传输层协议。该层共有两个协议传输数据：传输控制协议 TCP 和用户数据报协议 UDP。TCP 协议是面向连接的一个协议，可靠性高，费用也高；UDP 协议是提供最少服务和费用的传输层协议。

UDP 是最简单的传输层协议，它具有以下特点：

(1) 无连接：UDP 不基于连接来传输数据。

(2) 不可靠：UDP 的数据报发送时没有定序，所以 UDP 传送的数据是不可靠的，只有在应用层协议中增加超时重发和提供可靠服务。

(3) 提供应用层协议标识：UDP 报头有定义源应用层协议标识和目标应用层协议标识，这个在分析帧结构时再详细叙述。

(4) 提供 UDP 报的校验和：UDP 报头包含有整个 UDP 报（包括报头和有效负载）及伪报头的校验和，计算方法与 IP 数据报的校验和计算方法相同。

(5) 缓冲：UDP 协议不提供任何数据接收或发送的缓冲区。缓冲区应该由应用层协议提供。

(6) 分段：UDP 协议不提供分段传输方式，所以应用层协议要尽量发送小的包。

LwIP 的RAW API提供了较底层的UDP接口函数。这些函数与BSD标准的Sockets相比虽然编程较困难，但是它们生成的代码了更小，运行占用更少的内存，在RAM与CODE都是比较宝贵的嵌入式系统中，这通常我们的首选。况且，这些函数的编程思想是和BSD标准的Sockets相通的，都是基于"open-read-write-close"模型的。下面我们利用这些函数来实现使用UDP协议通信的数据收发。源代码如程序清单 3.1所示。

程序清单 3.1 UDP 服务器通信实例代码

```
void UDP_Receive(void *arg, struct udp_pcb *upcb, struct pbuf *p,struct ip_addr *addr, u16_t port)
{
    struct ip_addr destAddr = *addr;                    /*  获取远程主机  IP 地址          */
    if(p != NULL)                                       /*    如果收到的数据不为空         */
```

```
    {
    udp_sendto(upcb,p,&destAddr,port);                    /*  将收到的数据再发送出去          */
    pbuf_free(p);                                         /*  释放缓冲区数据                  */
    }
}

void UDP_Test(void)
{
    struct udp_pcb *UdpPcb;

    UdpPcb = udp_new();                                   /*  建立 UDP 通信的控制块(pcb)      */

    udp_bind(UdpPcb,IP_ADDR_ANY,1025);                    /*  绑定本地 IP 地址                */

    udp_recv(UdpPcb,UDP_Receive,NULL);                    /* 设置数据接收时的回调函数         */
}

int main()
{
    targetInit();                                         /*  初始化硬件                      */
    InitNic();                                            /*  初始化 LwIP 协议栈，包括软件    */
                                                          /*  和硬件                          */

    UDP_Test_Init();
    while(1)
    {
        ;
    }
}
```

这两个函数共同完成了 UDP 数据的收发。函数 UDP_Test()完成了以下几个功能。

- 建立用于通信的 udp_pcb。
- 绑定本地 IP 地址。
- 设置数据接收时的回调函数 UDP_Receive()。

函数 UDP_Receive()就是在数据接收时的回调函数。当有 UDP 数据接收到时，该函数被调用，并将接收到的数据存放到 pbuf p 中，同时获取远程主机的 IP 地址及端口号等信息，如果 p 不为空，说明数据被正确地接收，这时我们再将数据发送回去。

程序运行后，打开“TCP&UDP测试工具”，目标板的IP地址是“192.168.1.25”，在该IP地址的 1025 端口上建立一个UDP连接。然后设置定时发送一个字符串数据，我们可以在接

收框中看到接收回来的数据。如图 3.1所示。

图 3.1　UDP 服务器通信结果

3.2　UDP 客户端通信实例

本节将来完成一个UDP客户端的程序。客户端是指主动发送数据的一方，其实在上一节的例程中已经使用过UDP数据发送的函数，可能会发现它有些不寻常，不合于用户的习惯。udp_send()函数发送的数据存放在pbuf结构中，而BSD标准的Sockets的UDP发送数据的函数发送的都是字符型数组，即是一个字符串。这就使用户编程遇到了麻烦，必须对字符串数据进行一定的处理，将数据保存在pbuf结构中，才能够发送。程序实现代码如程序清单 3.2所示。

程序清单 3.2　UDP 客户端通信实例代码

```
void    Delay(unsigned long    ulVal)                    /*   利用循环产生一定的延时  */
{
      while ( --ulVal    !=    0 );
}
const static int8 UDPData[]="LwIP UDP 客户端在 Luminary 微控制器上的测试\r\n";
int main()
{
   targetInit();
   InitNic();

   struct udp_pcb *UdpPcb;
   struct ip_addr ipaddr;
```

```
  struct pbuf *p;

  p = pbuf_alloc(PBUF_RAW,sizeof(UDPData),PBUF_RAM);
  p->payload=(void *)UDPData;

  IP4_ADDR(&ipaddr,192,168,1,16);
  UdpPcb = udp_new();

  udp_bind(UdpPcb,IP_ADDR_ANY,1025);                    /*  绑定本地 IP 地址        */
  udp_connect(UdpPcb,&ipaddr,1025);                     /*  连接远程主机            */

  while(1)
  {
     udp_send(UdpPcb,p);
     Delay(1000000UL);
      Delay(1000000UL);
       Delay(1000000UL);
  }
}
```

本程序给出的是UDP客户端通信的主函数。程序运行后，首先，初始化硬件，初始化LwIP协议栈，之后就可以正常调查调用LwIP的RAW API函数了。UDP通信的建立和上节的一样，这里就不在多做说明，关键是对发送数据的处理，如何将数据放到pbuf变量中。要完成这个功能，首先要建立一个pbuf的结构变量。然后调用函数pbuf_alloc()为其分配内存空间，同时也要指定pbuf存放的位置。函数pbuf_alloc()的说明如表 3.1所示。调用该函数后，就可以将要发送的数据存放在pbuf结构中了，数据在pbuf中就是它的数据项p->payload。之后，就可以进行正常数据发送了，其它部分程序中都有详细的注释，也不在多做说明。需要注意的是，发送一次数据后需要一定的延时，总不能不停地发送数据。

表 3.1　函数 pbuf_alloc()

功能	分配一个指定类型(例如一个 PBUF_POOL 链表)的 pbuf
原型	struct pbuf *pbuf_alloc (pbuf_layer layer, u16_t size, pbuf_type type)
参数	layer：　通过包类型指定报头的大小 size：　pbuf->payload 的长度，即 pbuf 中实际存放数据的长度

	type： 该参数决定怎样以及在那个存储区分配 pbuf 结构，具体如下： PBUF_RAM：为 pbuf 分配一块相当大的缓冲区内存，它同时包含了协议头。 PBUF_ROM：没有为 pbuf 分配缓冲区内存甚至协议头。额外的头部信息会分配在 ROM pbuf 的链表的前面。假如一个内存区的使用和 ROM 非常地相似而是只读的，那么可以将 pbuf ROM 定义在该区。但是当内存是动态的，那就不能在该区定义 PBUF_ROM pbufs，而只能定义 PBUF_REF。 PBUF_REF：没有为 pbuf 分配缓冲区内存甚至协议头。假定该 pbuf 仅仅使用在单一的线程里面。如果 pbuf 在队列中，那么就应该以复制缓冲区的方式来访问它。 PBUF_POOL：作为一个链表来分配 pbuf，并且分配时应该在初始化函数 pbuf_init()中。
返回	pbuf： 分配了内存的 pbuf 结构

程序运行后，打开“TCP&UDP测试工具”，目标板的IP地址是“192.168.1.25”，在该IP地址的 1025 端口上建立一个UDP连接。点击连接按钮后，在接收数据框中，可看到发送的数据，如图 3.2所示。

图 3.2 UDP 客户端通信结果

3.3 WEB 服务器实例

传输层的 TCP 协议全称是传输控制协议，从 TCP/IP 这个名字我们就可以知道 TCP 的分量。TCP 是基于 IP 数据帧的传输，提供可靠的数据传输服务。TCP 具有以下特点：

(1) 面向连接：TCP 协议规定，在进行数据传输之前，两个节点必须使用 TCP 连接的建立过程进行连接，建立连接成功后，再进行数据传输。终止连接也要使用 TCP 连接中断过程关闭连接，如何连接和中断我们在以后再详细说明。

(2) 双向传输：TCP 协议中，每有个连接都 2 个逻辑管道，一收一发。数据可以同时进行收和发，TCP 报头包含传出数据和确认输入数据的序列号。

(3) 可靠传输：TCP 协议规定，在传输数据时，要按顺序发送数据，并得到接收方的确认，没有得到确认的数据将重发，接收方接收到重复包将丢弃，失序包将被还原为正确的序

列。TCP 校验字提供比特级的完整性校验。

(4) 数据字节流：在 TCP 的输入和输出逻辑管道上传输的数据被认为是连续的字节流，TCP 报头的序列号和确认号都是以字节为单位确定的。TCP 不知道所传输的字节流的内容是什么，以及在那里起始，在那里结束，对字节流数据的分析只能通过应用层的协议进行分析。

(5) 流控制：TCP 协议中还规定了数据传输的流量控制，防止堵塞。在数据传输的双方都有接收缓冲区，如果一方接收缓冲区满了，另一方将不再发送，直到缓冲区有空余的空间。而双方的接收缓冲区都是独立的，永远也不会溢出。

(6) 应用层数据分段：TCP 建立连接时，双方都交换可接收的最大段，如果接收到 ICMP 的“路径最大传输单位”MTU 消息，能自动调整 TCP 最大段的大小。

(7) 一对一传输：TCP 协议实现一对一通讯服务。

LwIP 的RAW API提供了较底层的TCP接口函数。这些函数与BSD标准的Sockets相比虽然编程较困难，但是它们生成的代码了更小，运行占用更少的内存，在RAM与CODE都是比较宝贵的嵌入式系统中，这通常我们的首选。况且，这些函数的编程思想是和BSD标准的Sockets相通的，都是基于"open-read-write-close"模型的。本小节我们利用这些函数来建立一个WEB服务器。源代码如程序清单 3.3所示。

程序清单 3.3 WEB 服务器实例代码

```
/*************************TCP 测试的 WEB 服务器建立网页与服务器声明**********************/
const static uint8 indexdata[]="<html>\
                                <head><title>广州致远电子</title></head>\
                                <body>\
                                 这里是 LwIP TCP 在 Luminary Cortex-M3 上的测试！\
                                </body>\
                                </html>";
const static uint8 http_html_hdr []="HTTP/1.1 200 OK\r\nContent-type: text/html\r\n\r\n";

/******************** 这是一个回调函数，当一个 TCP 段到达这个连接时会被调用*****************/
static err_t http_recv(void *arg, struct tcp_pcb *pcb,struct pbuf *p,err_t err)
{
 if(p != NULL)
 {
     tcp_write(pcb,http_html_hdr,sizeof(http_html_hdr),0);          /* 发送 http 协议头部信息      */
     tcp_write(pcb,indexdata,sizeof(indexdata),0);                  /* 发送 http 网页信息          */

     pbuf_free(p);                                                  /* 释放该 TCP 段               */
 }
  tcp_close(pcb);                                                   /* 关闭这个连接                */
  err = ERR_OK;
  return err;
```

```
}

/**********************这是一个回调函数，当一个连接已经接受时会被调用**********************/
static err_t http_accept(void *arg,struct tcp_pcb *pcb,err_t err)
{
   tcp_setprio(pcb, TCP_PRIO_MIN);                          /*   设置回调函数优先级，当     */
                                                            /*   存在几个连接时特别重要，   */
                                                            /*   此函数必须调用*/
   tcp_recv(pcb,http_recv);                                 /*   设置 TCP 段到时的回调函数  */

   err = ERR_OK;
   return err;
}
void http_init(void)
{
   struct tcp_pcb *pcb;

   pcb = tcp_new();                                         /*  建立通信的 TCP 控制块(pcb)      */
   tcp_bind(pcb,IP_ADDR_ANY,80);                            /*  绑定本地 IP 地址和端口号        */
   pcb = tcp_listen(pcb);                                   /*  进入监听状态                    */
   tcp_accept(pcb,http_accept);                             /*  设置有连接请求时的回调函数      */
}

int main()
{
   targetInit();
   InitNic();

   http_init();
   while(1)
   {
      ;
   }
}
```

上面这几个函数共同完成了WEB服务器的功能。函数http_init()完成服务器的初始化，进入监听状态，注册连接请求接收到时的回调函数。其余两个函数http_accept()和http_recv()都是回调函数。它们在不同的状态下调用。函数的具体注释程序清单 3.3给出的比较详细，这里就不在多做说明了。

程序运行后，服务器在初始化以后进入监听状态，这时我们可以在PC机上通过IE浏览器查看网页内容。打开IE浏览器，输入IP地址：“192.168.1.25”，即可看到网页内容，如图 3.3 所示。

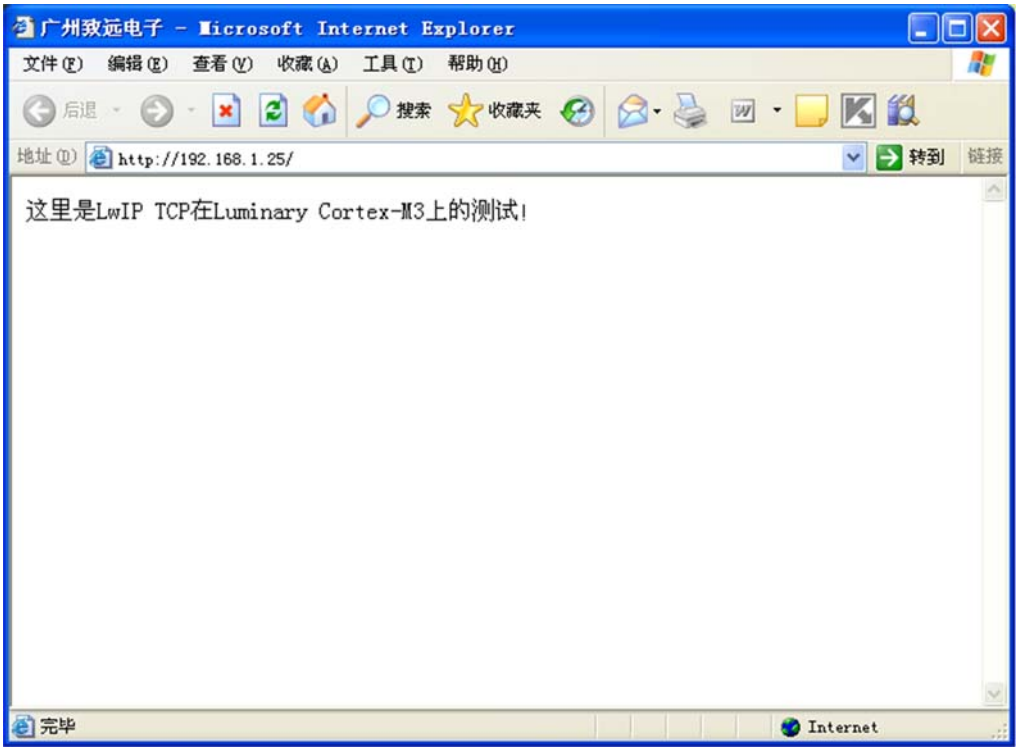

图 3.3 WEB 服务器效果图

3.4 TCP 客户端通信实例

现在，再来完成TCP客户端的程序编写。客户端是指主动发送数据的一方，程序运行后，客户机应该请求与服务器的连接，连接成功后就可以进行数据收发了。在第二章介绍LwIP的RAW API函数时已经介绍过，使用RAW API编程是基于回调函数的思想的，同时在介绍RAW API函数的时候也有几个函数是专门用来注册回调函数的。同样，这里的编程也是利用回调函数的。具体实现代码如程序清单 3.4所示。

程序清单 3.4 TCP 客户端通信代码

```
const static uint8 TCP_TestData[]="This is LwIP TCP Client  在 Luminary Cortex-M3 上的测试！ \r\n";
void TCP_Client_Init();

void    Delay(unsigned long    ulVal)                          /*    利用循环产生一定的延时              */
{
      while ( --ulVal    !=    0 );
}

int main()
{
   targetInit();
```

```
   InitNic();

   while(1)
   {
      TCP_Client_Init();
      Delay(1000000UL);
      Delay(1000000UL);
      Delay(1000000UL);
    }
}

/******* 这是一个回调函数，当 TCP 客户端请求的连接建立时被调用********/
err_t TcpCli_Connected(void *arg,struct tcp_pcb *pcb,err_t err)
{
    tcp_write(pcb,TCP_TestData,sizeof(TCP_TestData),0);          /*   发送数据                          */

    tcp_close(pcb);

    return ERR_OK;
}

void TCP_Client_Init()
{
   struct tcp_pcb *Clipcb;
   struct ip_addr ipaddr;

   IP4_ADDR(&ipaddr,192,168,1,16);

   Clipcb = tcp_new();                                    /*   建立通信的 TCP 控制块(Clipcb)     */
   tcp_bind(Clipcb,IP_ADDR_ANY,1026);                     /*   绑定本地 IP 地址和端口号          */

   tcp_connect(Clipcb,&ipaddr,1026,TcpCli_Connected);
}
```

程序清单中，程序运行后，进入主函数，先调用函数 targetInit()初始化设备，主要完成了系统时钟的设置，初始化系统时钟定时器供 LwIP 的周期性事件使用。随后，调用函数 InitNic()，该函数初始化了 LwIP(包括硬件以太网控制器及软件 TCP/IP 协议栈)，然后就可以调用 LwIP 的 RAW API 函数编程了。在函数 TCP_Client_Init()中，建立了一个 TCP 连接，并向 IP 地址为“192.168.1.16”的远程主机的 1026 号端口发出连接请求，同时注册了一个

回调函数 TcpCli_Connected()，当连接正确建立后，该函数将被调用。在回调函数 TcpCli_Connected()中向连接的主机发送数据，发送完毕后关闭本次连接。在主函数中，延时一段时间以后，再次建立连接，如此周而复始地工作。

本程序的测试需要一个上位机软件用来查看发送出去的数据。该软件实际上就是一个 TCP 服务器，它的工作过程为，程序运行后，建立 TCP 通信的 Sockets，并使其进入监听状态，等接收到连接请求时，同意建立连接，然后就开始接收数据并同时在一个窗口中将其显示出来，之后就关闭连接同时让你再次进入监听状态，等待下次连接。如此，周而复始地工作。笔者到处也找不到这样的测试软件，于是就在 Visual C++ 6.0 下，建立一个 MFC 工程，编写了这样的一个测试软件，完全满足了测试本程序的目的。关于上位机的代码，这里不在给出，总之，该软件对于测试 TCP 客户端来说很好用，笔者会一并给出。

在测试时，先运行上位机软件，其实先运行那个也没有关系，但是通常大家都会认为服务器的一方总是最先启动的。启动后，先输入端口号后在点击开启服务器，注意一定要先输入端口号，如果直接点击开启服务器将会弹出一个警告对话框，提示用户先输入端口号。成功开启服务器后，该按钮将变灰，防止再次开启。

服务器开启后，再运行下位机中的程序。可以看到运行结果如程序清单 3.4所示。

图 3.4　TCP 客户端通信结果