TCP/IP 网络组件 Lwip 之 TCP Server

RealTouch 评估板 RT-Thread 入门文档

版本号：1.0.0

日期：2012/8/29

修订记录

日期	作者	修订历史
2012/8/29	bloom5	创建文档

实验目的

- 快速了解 Lwip 组件。
- 了解使用 TCP 协议进行网络通信的方法和过程。

硬件说明

本实验使用 RT-Thread 官方的 Realtouch 开发板作为实验平台。涉及到的硬件主要为

- RJ45 接口，作为网络连接的需要，我们需要用网线将 Realtouch 和目标机连接起来，具体请参见《Realtouch 开发板使用手册》
- 串口 3，作为 rt_kprintf 输出，需要连接 JTAG 扩展板

实验原理及程序结构

实验设计

本实验设计为在 RealTouch 开发板上运行 TCP Server，PC 机作为 TCP Client，两机进行网络通信。

源程序说明

系统依赖

在 rtconfig.h 中需要开启

- #define RT_USING_HEAP

此项可选，开启此项可以创建动态线程和动态信号量，如果使用静态线程和静态信号量，则此项不是必要的

- #define RT_USING_LWIP

此项必须，本实验使用 LWIP 组件，因此需要开启此项

- #define RT_USING_CONSOLE

此项必须，在开始过程中仍需通过串口进行显示相关的工作

主程序说明

关于本实验，在初始化线程中完成了网络硬件的初始化，lwip 初始化，然后启动了 TCP Server。

```
  void rt_init_thread_entry(void* parameter)
  {
  #ifdef RT_USING_LWIP
    /* initialize eth interface */
    rt_hw_stm32_eth_init();
  #endif

  #ifdef RT_USING_COMPONENTS_INIT
     /* initialization RT-Thread Components */
    rt_components_init();
  #endif

    rt_platform_init();
     /* do some thing here. */
    tcpserv();
  }

  int rt_application_init()
  {
     rt_thread_t init_thread;

     init_thread = rt_thread_create("init",
                                rt_init_thread_entry, RT_NULL,
                                2048, 8, 20);


     if (init_thread != RT_NULL)
        rt_thread_startup(init_thread);

     return 0;
  }
```

Tcpsrv 所有的操作均在 tcpserver.c 中的 tcpsrv()函数中完成，源码配有详尽中文注释，可以帮助进一步你了解其工作机制。

```
  static const char send_data[] = "This is TCP Server from
RT-Thread."; /* 发送用到的数据 */
  void tcpserv(void *parameter)
  {
     char *recv_data; /* 用于接收的指针，后面会做一次动态分配以请求可用
内存 */
     rt_uint32_t sin_size;
     int sock, connected, bytes_received;
```

```
    struct sockaddr_in server_addr, client_addr;
    rt_bool_t stop = RT_FALSE; /* 停止标志 */

    recv_data = rt_malloc(1024); /* 分配接收用的数据缓冲 */
    if (recv_data == RT_NULL)
    {
        rt_kprintf("No memory\n");
        return;
    }

    /* 一个 socket 在使用前，需要预先创建出来，指定 SOCK_STREAM 为 TCP 的
socket */
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        /* 创建失败的错误处理 */
        rt_kprintf("Socket error\n");

        /* 释放已分配的接收缓冲 */
        rt_free(recv_data);
        return;
    }

    /* 初始化服务端地址 */
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5000); /* 服务端工作的端口 */
    server_addr.sin_addr.s_addr = INADDR_ANY;
    rt_memset(&(server_addr.sin_zero), 8,
sizeof(server_addr.sin_zero));

    /* 绑定 socket 到服务端地址 */
    if (bind(sock, (struct sockaddr *)&server_addr, sizeof(struct
sockaddr)) == -1)
    {
        /* 绑定失败 */
        rt_kprintf("Unable to bind\n");

        /* 释放已分配的接收缓冲 */
        rt_free(recv_data);
        return;
    }

    /* 在 socket 上进行监听 */
    if (listen(sock, 5) == -1)
    {
```

```
        rt_kprintf("Listen error\n");

        /* release recv buffer */
        rt_free(recv_data);
        return;
    }

    rt_kprintf("\nTCPServer Waiting for client on port
5000...\n");
    while (stop != RT_TRUE)
    {
        sin_size = sizeof(struct sockaddr_in);

        /* 接受一个客户端连接 socket 的请求，这个函数调用是阻塞式的 */
        connected = accept(sock, (struct sockaddr *)&client_addr,
&sin_size);
        /* 返回的是连接成功的 socket */

        /* 接受返回的 client_addr 指向了客户端的地址信息 */
        rt_kprintf("I got a connection from (%s , %d)\n",
                inet_ntoa(client_addr.sin_addr),
ntohs(client_addr.sin_port));

        /* 客户端连接的处理 */
        while (1)
        {
            /* 发送数据到 connected socket */
            send(connected, send_data, strlen(send_data), 0);

            /* 从 connected socket 中接收数据，接收 buffer 是 1024 大小，
但并不一定能够收到 1024 大小的数据 */
            bytes_received = recv(connected, recv_data, 1024, 0);
            if (bytes_received <= 0)
            {
                /* 接收失败，关闭这个 connected socket */
                lwip_close(connected);
                break;
            }

            /* 有接收到数据，把末端清零 */
            recv_data[bytes_received] = '\0';
            if (strcmp(recv_data , "q") == 0 || strcmp(recv_data ,
"Q") == 0)
            {
```

```
                /* 如果是首字母是 q 或 Q，关闭这个连接 */
                lwip_close(connected);
                break;
            }
            else if (strcmp(recv_data, "exit") == 0)
            {
                /* 如果接收的是 exit，则关闭整个服务端 */
                lwip_close(connected);
                stop = RT_TRUE;
                break;
            }
            else
            {
                /* 在控制终端显示收到的数据 */
                rt_kprintf("RECIEVED DATA = %s \n" , recv_data);
            }
        }
    }

    /* 退出服务 */
    lwip_close(sock);

    /* 释放接收缓冲 */
    rt_free(recv_data);

    return ;
}
```

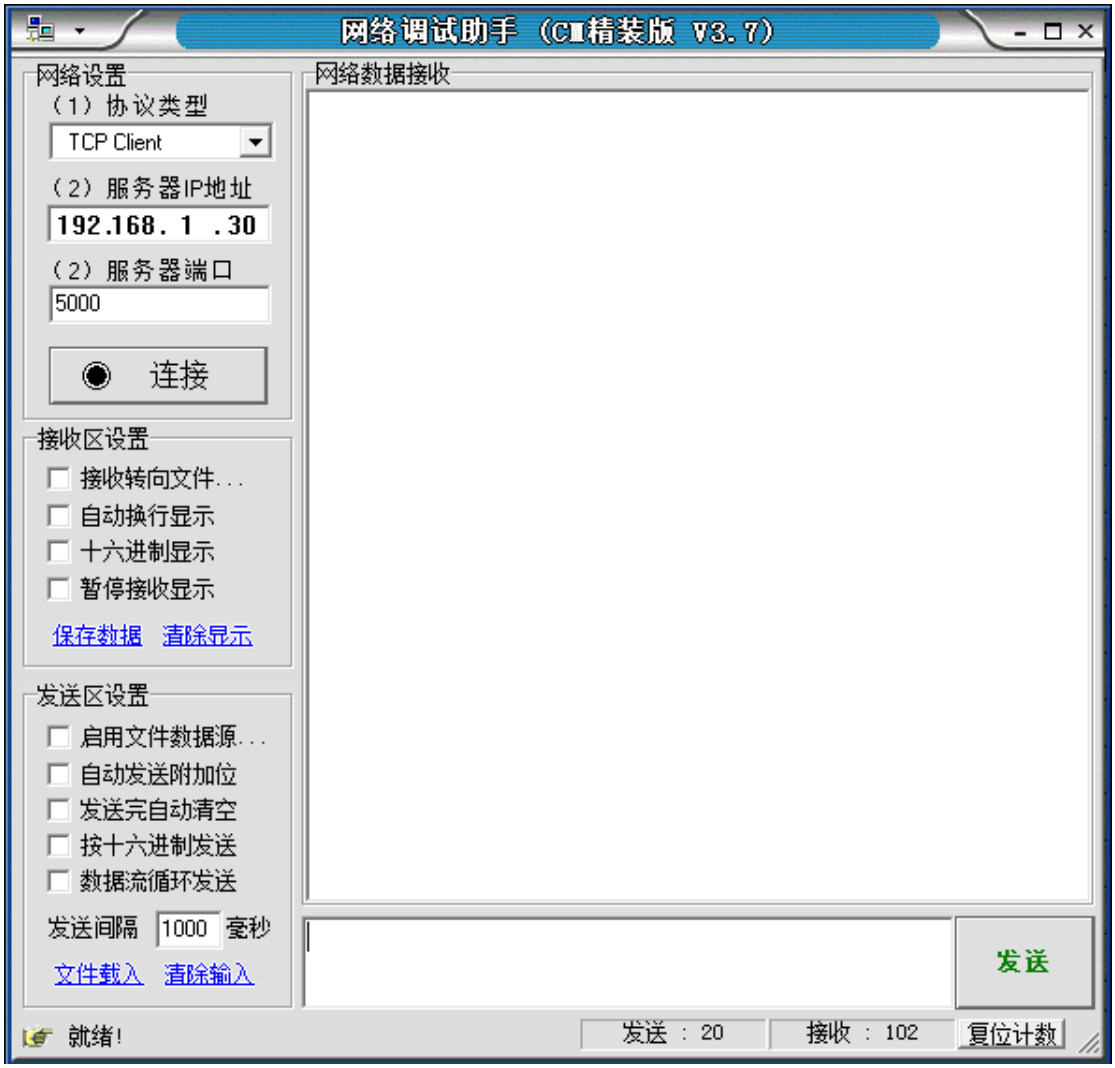
编译调试及观察输出信息

编译请参见《RT-Thread 配置开发环境指南》完成编译烧录，参考《Realtouch 开发板使用手册》完成硬件连接，连接好串口线，连上网线。

运行后可以看到串口有如下的信息：

```
 \ | /
 - RT -     Thread Operating System
  / | \     1.1.0 build Aug 29 2012
  2006 - 2012 Copyright by rt-thread team
 TCP/IP initialized!
 finsh>>
 TCPServer Waiting for client on port 5000...
```

然后在启动 PC 机上的 TCP Client 客户端，将其与 TCP Server 连接。



如果连接成功，会收到 TCP Server 端发来第一条信息。

因为连接是双向的，客户端也可以向 TCP Server 发送消息。

于是在串口上可以看到：

```
I got a connection from (192.168.1.11 , 52925)
RECIEVED DATA = yeah! i got a message
```

结果分析

通过以上内容的实际操作，实现了 TCP Server 与 TCP Client 的连接与网络通信。