实时操作系统μC/OS II 下 TCP/IP 协议栈的实现

摘要： 结合 ez80 和 ARM7 两种系统上的具体实现，说明了如何在嵌入式实时操作系统μC/OSII 上移植实现 LwIP 这套 TCP/IP 协议栈，使μC/OS II 成为支持网络的 RTOS。

关键词： μC/OS II，TCP/IP，LwIP，网络设备驱动

引言

随着嵌入式系统与网络的日益结合，在嵌入式实时操作系统中引入 TCP/IP 协议栈，以支持嵌入式设备接入网络，成为嵌入式领域重要的研究方向。μC/OS II 是近年来发展迅速的一个开放源码实时操作系统，但它只是一个实时的任务调度及通信内核，缺少对外围设备和接口的支持，如没有文件系统、网络协议、图形界面。笔者在多个嵌入式项目的开发过程中，以开源 TCP/IP 协议栈 LwIP 为基础，给μC/OS II 加上了网络支持。下面就以μC/OS II ＋LwIP 分别在 8 位 MCU ez80 和 32 位 MCU ARM7TDMI 上的实现为例进行说明。

需要说明的是，笔者使用的 ez80 系统是 Zilog 公司的 ez80190 开发板，自带网络芯片。而 ARM7 系统是使用笔者参与开发的 Skyeye，一个基于 GDB 的 ARM7TDMI 指令级软件仿真器。Skyeye 小组最近为 Skyeye 加上了软件模拟的 Ne2k 兼容网络芯片，可以运行带网络支持的μcLinux 和μC/OS II。以下的全部相关程序和代码都可以在 Skyeye 网站（hpclab.cs.tsinghua.edu.cn/~skyeye/）下载。

1 基于μC/OS II 的网络平台概述

嵌入式操作系统μC/OS II 是一个公开源代码的占先式多任务的微内核 RTOS，其性能和安全性可以与商业产品竞争。μC/OS II 的特点可以概括为以下几个方面：公开源代码，代码结构清晰、明了，注释详尽，组织有条理，可移植性好。可裁剪，可固化。内核属于抢占式，最多可以管理 60 个任务。μC/OS II 自 1992 年的第一版（μC/OS）以来已经有好几百个应用，是一个经实践证明好用且稳定可靠的内核。目前国内对μC/OS II 的研究和应用都很多。

TCP/IP 是 Internet 的基本协议，以其实用性、高效性已经成为事实上的工业标准。嵌入式设备要与 Internet 网络直接交换信息，就必须支持 TCP/IP 协议。目前嵌入式设备上 TCP/IP 方案有很多种，但面向低端应用的开源嵌入式网络平台还很少见。

μC/OS II 是一个富有开放色彩的 RTOS，只要买一本书就可获得源代码，对学校和教育的使用完全免费，商业应用的费用相对也很低。但是它目前的一些第三方 TCP/IP 支持都是完全商业化的，用户需要付费才能获得，很少给出源代码，这影响了μC/OS II 的研究和推广。通过把开放源代码的 TCP/IP 协议栈 LwIP 移植到μC/OS II 上来，就获得了一套可免费研究、学习的嵌入式网络软件平台。系统示意图如图 1：

图 1 μC/OS II＋LwIP 系统示意图

2　开源 TCP/IP 协议栈 LwIP 简介

LwIP 是瑞士计算机科学院（Swedish Institute of Computer Science）的 Adam Dunkels 等开发的一套用于嵌入式系统的开放源代码 TCP/IP 协议栈。LwIP 的含义是 Light Weight(轻型)IP 协议。LwIP 可以移植到操作系统上，也可以在无操作系统的情况下独立运行。LwIP TCP/IP 实现的重点是在保持 TCP 协议主要功能的基础上减少对 RAM 的占用，一般它只需要几十 K 的 RAM 和 40K 左右的 ROM 就可以运行，这使 LwIP 协议栈适合在低端嵌入式系统中使用。

LwIP 的特性如下：

(1)　支持多网络接口下的 IP 转发

(2)　支持 ICMP 协议

(3)　包括实验性扩展的的 UDP（用户数据报协议）

(4)　包括阻塞控制，RTT 估算和快速恢复和快速转发的 TCP（传输控制协议）

(5)　提供专门的内部回调接口（Raw API）用于提高应用程序性能

(6)　可选择的 Berkeley 接口 API（多线程情况下）

我们目前使用的是 LwIP 的最新稳定版 V0.5.3。有关 LwIP 的详细内容，可以参考其代码和网站上的文档。

4　LwIP 在μC/OS II 下的实现

4.1 概述

LwIP 协议栈在设计时就考虑到了将来的移植问题，因此把所有与硬件、OS、编译器相关的部份独立出来，放在/src/arch 目录下。因此 LwIP 在μC/OS II 上的实现就是修改这个目录下的文件，其它的文件一般不应该修改。下面分几部份分别说明相应文件的实现原理和过程。具体的代码限于篇幅没有给出，Skyeye 网站上有完整的代码和说明。

4.2 与 CPU 或编译器相关的 include 文件

/src/arch/include/arch 目录下 cc.h、cpu.h、perf.h 中有一些与 CPU 或编译器相关的定义，如数据长度，字的高低位顺序等。这应该与用户实现μC/OS II 时定义的数据长度等参数是一致的。

```
#define BYTE_ORDER LITTLE_ENDIAN   //ARM7 默认为小端存储系统
//数据类型长度的定义
typedef unsigned char     u8_t;
typedef signed char       s8_t;
typedef unsigned short    u16_t;
typedef signed short      s16_t;
typedef unsigned int      u32_t;
typedef signed int        s32_t;
```

此外还有一点：一般情况下 C 语言的结构体 struct 是 4 字节对齐的，但是在处理数据包的时候，LwIP 使用的是通过结构体中不同数据的长度来读取相应的数据的，所以，一定要

在定义 struct 的时候使用_packed 关键字，让编译器放弃 struct 的字节对齐。LwIP 也考虑到了这个问题，所以，在它的结构体定义中有几个 PACKED_FIELD_xxx 宏，默认的时候这几个宏都是空的，可以在移植的时候添加不同的编译器所对应的_packed 关键字。比如在 Skyeye（ARM7）上对应 gcc 编译器的定义：

```
#define PACK_STRUCT_FIELD(x) x __attribute__((packed))
#define PACK_STRUCT_STRUCT __attribute__((packed))
#define PACK_STRUCT_BEGIN
#define PACK_STRUCT_END
```

4.3 sys_arch 操作系统相关部份

sys_arch.[ch]中的内容是与 OS 相关的一些结构和函数，主要可以分为四个部份：

(1) sys_sem_t 信号量

LwIP 中需要使用信号量通信，所以在 sys_arch 中应实现信号量结构体和处理函数：

```
struct  sys_sem_t
sys_sem_new（）          //创建一个信号量结构
sys_ sem _free（）       //释放一个信号量结构
sys_ sem _signal（）     //发送信号量
sys_ arch_sem _wait（）  //请求信号量
```

由于μC/OSII 已经实现了信号量 OS_EVENT 的各种操作，并且功能和 LwIP 上面几个函数的目的功能是完全一样的，所以只要把μC/OSII 的函数重新包装成上面的函数，就可以直接使用了。

(2) sys_mbox_t 消息

LwIP 使用消息队列来缓冲、传递数据报文，因此要在 sys_arch 中实现消息队列结构 sys_mbox_t，以及相应的操作函数：

```
sys_mbox_new（）          //创建一个消息队列
sys_mbox_free（）         //释放一个消息队列
sys_mbox_post（）         //向消息队列发送消息
sys_arch_mbox_fetch（）   //从消息队列中获取消息
```

μC/OSII 同样实现了消息队列结构 OSQ 及其操作，但是μC/OSII 没有对消息队列中的消息进行管理，因此不能直接使用，必须在μC/OSII 的基础上重新实现。为了实现对消息的管理，我们定义了以下结构：

```
typedef struct {
    OS_EVENT*     pQ;
    void* pvQEntries[MAX_QUEUE_ENTRIES];
} sys_mbox_t;
```

在以上结构中，包括 OS_EVENT 类型的队列指针（pQ）和队列内的消息（pvQEntries）两部分，对队列本身的管理利用μC/OSII 自己的 OSQ 操作完成，然后使用μC/OSII 中的内存管理模块实现对消息的创建、使用、删除回收，两部分综合起来形成了 LwIP 的消息队列功能。

(3) sys_arch_timeout 函数

LwIP 中每个与外界网络连接的线程都有自己的 timeout 属性，即等待超时时间。这个属性表现为每个线程都对应一个 sys_timeout 结构体队列，包括这个线程的 timeout 时间长度，以及超时后应调用的 timeout 函数，该函数会做一些释放连接，回收资源的工作。如果一个

线程对应的 sys_timeout 为空（NULL），说明该线程对连接做永久的等待。

timeout 结构体已经由 LwIP 自己在 sys.h 中定义好了，而且对结构体队列的数据操作也由 LwIP 负责，我们所要实现的是如下函数：

struct sys_timeouts * sys_arch_timeouts(void)

这个函数的功能是返回目前正处于运行态的线程所对应的 timeout 队列指针。timeout 队列属于线程的属性，因此是 OS 相关的函数，只能由用户实现。

(4) sys_thread_new 创建新线程

LwIP 可以是单线程运行，即只有一个 tcpip 线程（tcpip_thread），负责处理所有的 tcp/ucp 连接，各种网络程序都通过 tcpip 线程与网络交互。但 LwIP 也可以多线程运行，以提高效率，降低编程复杂度。这时就需要用户实现创建新线程的函数：

void sys_thread_new(void (* thread)(void *arg), void *arg);

在μC/OS II 中，没有线程（thread）的概念，只有任务（Task）。它已经提供了创建新任务的系统 API 调用 OSTaskCreate，因此只要把 OSTaskCreate 封装一下，就可以实现 sys_thread_new。需要注意的是 LwIP 中的 thread 并没有μC/OS II 中优先级的概念，实现时要由用户事先为 LwIP 中创建的线程分配好优先级。

4.4 lib_arch 中库函数的实现

LwIP 协议栈中用到了 8 个外部函数，这些函数通常与用户使用的系统或编译器有关，因此留给用户自己实现。如下：

```
u16_t htons(u16_t n);        //16 位数据高低字节交换
u16_t ntohs(u16_t n);
u32_t htonl(u32_t n);        //32 位数据大小头对调
u32_t ntohl(u32_t n);
int strlen(const char *str);     //返回字符串长度
int strncmp(const char *str1, const char *str2, int len);   //字符串比较
void bcopy(const void *src, void *dest, int len);   //内存数据块之间的互相拷贝
void bzero(void *data, int n);        //内存中指定长度的数据块清零
```

前四个函数通常由用户自己实现。Skyeye（ARM7）中，由于使用了 gcc 编译器，gcc 的 lib 库里已经有了后四个函数。而 ez80 的编译器函数库中缺少 bcopy 和 bzero 两个，需要自己编写。用户在其它 CPU 上实现时应根据自己的编译器来决定。

4.5 网络设备驱动程序

ez80 开发板自带的网络芯片为 RealTek 的 8019as 芯片，这是 ISA 10BASE－T 的以太网芯片，与 Ne2k 兼容。而我们在 AT91 模拟器 Skyeye 中所仿真的网络芯片也是 Ne2k，所以目前实现的网络设备驱动是针对 Ne2k 的，其它类型的网络芯片驱动可以在 LwIP 的网站上找到。LwIP 的网络驱动有一定的模型，/src/netif/ethernetif.c 文件即为驱动的模板，用户为自己的网络设备实现驱动时应参照此模板。

在 LwIP 中可以有多个网络接口，每个网络接口都对应了一个 struct netif，这个 netif 包含了相应网络接口的属性、收发函数。LwIP 调用 netif 的方法 netif->input()及 netif->output()进行以太网 packet 的收、发等操作。在驱动中主要做的，就是实现网络接口的收、发、初始化以及中断处理函数。驱动程序工作在 IP 协议模型的网络接口层，它提供给上层（IP 层）

的接口函数如下：

```
//网卡初始化函数
void ethernetif_init(struct netif *netif)
//网卡接收函数，从网络接口接收以太网数据包并把其中的 IP 报文向 IP 层发送
//在中断方式下由网卡 ISR 调用
void ethernetif_input(struct netif *netif)
//网卡发送函数，给 IP 层传过来的 IP 报文加上以太网包头并通过网络接口发送
err_t ethernetif_output(struct netif *netif, struct pbuf *p, struct ip_addr *ipaddr)
//网卡中断处理函数 ISR
void ethernetif_isr(void);
```

以上的函数都可以分为协议栈本身的处理和对网络接口硬件的操作两部份，但硬件操作是对上层屏蔽的，具体参见 RTL8019as、DM9008 等 Ne2k 网络芯片的数据手册。驱动程序可以到 Skyeye 或 LwIP 的网站下载。

5 应用实例的建立和测试

做完上面的移植修改工作以后，就可以在μC/OSII 中初始化 LwIP，并创建 TCP 或 UDP 任务进行测试了。这部份完全是 C 语言的实现，因此这部份在 ez80 和 ARM7 上基本都是一样的。值得注意的是 LwIP 的初始化必须在μC/OSII 完全启动之后也就是在任务中进行，因为它的初始化用到了信号量等 OS 相关的操作。关键部份的代码和说明如下：

```
main(){
    OSInit();
    OSTaskCreate(lwip_init_task, &LineNo11, &lwip_init_stk[TASK_STK_SIZE-1], 0);
    OSTaskCreate(usr_task,&LineNo12,&usr_stk[TASK_STK_SIZE-1],1);
    OSStart();
}
```

主程序中创建了 lwip_init_task 初始化 LwIP 任务（优先级 0）和 usr_task 用户任务（优先级 1）。lwip_init_task 任务中除了初始化硬件时钟和 LwIP 之外，还创建了 tcpip_thread（优先级 5）和 tcpecho_thread（优先级 6）。实际上 tcpip_thread 才是 LwIP 的主线程，多线程的 Berkley API 也是基于这个线程实现的，即上面的 tcpecho_thread 线程也要依靠 tcpip_thread 线程来与外界通信，这样做的好处是编程简单，结构清晰。

实用 Berkley API 实现的 tcpecho_thread 是一个 TCP echo 服务器，监听 7 号端口，程序框架如下：

```
void tcpecho_thread(void *arg){
  conn = netconn_new(NETCONN_TCP);  //创建新的连接标识
  netconn_bind(conn, NULL, 7);      //绑定到 7 号端口
  netconn_listen(conn);             //开始监听端口
  while(1){
    newconn = netconn_accept(conn);     //接收外部到来的连接
    buf = netconn_recv(newconn)         //获取数据
    …….                                 //处理数据
    netconn_write(newconn, data, len, NETCONN_COPY); //发送数据
    netconn_delete(newconn);            //释放本次连接
    }
```

}

编译运行后，用 ping ip 地址命令可以得到 ICMP reply 响应。用 telnet ip 地址 7（登录 7 号端口）命令可以看到 echo server 的回显效果。说明 ARP、ICMP、IP、TCP 协议都已正确运行。

参考文献

1. JEAN J.LABROSSE，邵贝贝译. μC/OS－II ——源码公开的实时嵌入式操作系统. 中国电力出版社
2. Adam Dunkels. LwIP source code.
3. Adam Dunkels. sys_arch.txt in LwIP source code.