LwIP 移植和使用

本手册基于 lwip-1.4.x 编写，本人没有移植过 1.4.0 之前的版本，更早的版本或许有差别。
如果看官发现问题欢迎联系<QQ: 937431539　email: 937431539@qq.com>

LwIP 官网是：http://savannah.nongnu.org/projects/lwip/
你可以从这里获取源代码。当然也可以从 Git 获取源代码：
```
git clone git://git.savannah.nongnu.org/lwip.git
```

LwIP 以 BSD 协议发布源代码，我们可以自由的使用，修改，发布或不发布源代码。

附件中有我移植的文件，可以用来参考。祝你移植顺利。

移植

1)新建几个头文件：

```
include/lwipopts.h       // lwip 配置文件
include/arch/cc.h        // 平台相关。类型定义,大小端设置,内存对齐等
include/arch/perf.h      // 平台相关的性能测量实现(没用)
include/arch/sys_arch.h  // RTOS 抽象层。信号量，mbox 等类型定义，函数声明
```

lwipopts.h　　// lwip 配置文件，详见附件

cc.h　　//类型定义,大小端设置,内存对齐等

```
#ifndef __CC_H__
#define __CC_H__

#include <stdint.h>

/* Types based on stdint.h */
typedef uint8_t            u8_t;
typedef int8_t             s8_t;
typedef uint16_t           u16_t;
typedef int16_t            s16_t;
typedef uint32_t           u32_t;
typedef int32_t            s32_t;
typedef uintptr_t          mem_ptr_t;

/* Define (sn)printf formatters for these lwIP types */
#define U16_F "hu"
#define S16_F "hd"
#define X16_F "hx"
#define U32_F "lu"
#define S32_F "ld"
#define X32_F "lx"
#define SZT_F "uz"

/* 选择小端模式 */
#define BYTE_ORDER LITTLE_ENDIAN

/* Use LWIP error codes */
#define LWIP_PROVIDE_ERRNO

/* 内存对齐 */
#if defined(__arm__) && defined(__ARMCC_VERSION)
    /* Keil uVision4 tools */
```

```
    #define PACK_STRUCT_BEGIN __packed
    #define PACK_STRUCT_STRUCT
    #define PACK_STRUCT_END
    #define PACK_STRUCT_FIELD(fld) fld
    #define ALIGNED(n)  __align(n)
#endif
```

perf.h　　// 两个宏定义为空即可

```
#ifndef __PERF_H__
#define __PERF_H__

#define PERF_START    /* null definition */
#define PERF_STOP(x)  /* null definition */

#endif /* END __PERF_H__ */
```

sys_arch.h

RTOS 抽象层的类型定义，函数声明，详细内容见 doc/sys_arch.h

2)建立 RTOS 抽象层文件：

port/sys_arch.c　　// RTOS 抽象层实现

为了屏蔽不同 RTOS 在信号量，互斥锁，消息，任务创建等 OS 原语使用上的差别，lwip 构造了一个 RTOS 的抽象层，规定了 OS 原语的数据类型名称和对应方法名称。我们要做的就是根据所用 RTOS 的 api 去实现这些原语。

比如移植 lwip 到 raw-os 上，信号量的移植：

类型定义，宏定义在 sys_arch.h 中

```
/* HANDLE is used for sys_sem_t but we won't include windows.h */
struct _sys_sem {
  RAW_SEMAPHORE *sem;
};

typedef struct _sys_sem sys_sem_t;      // sys_sem_t 是 lwip 的信号量类型名
#define SYS_SEM_NULL NULL
#define sys_sem_valid(sema) (((sema) != NULL) && ((sema)->sem != NULL))
#define sys_sem_set_invalid(sema) ((sema)->sem = NULL)

err_t sys_sem_new(sys_sem_t *sem, u8_t count)
{
    RAW_SEMAPHORE *semaphore_ptr = 0;

    if (sem == NULL)
    {
        RAW_ASSERT(0);
    }

    semaphore_ptr = port_malloc(sizeof(RAW_SEMAPHORE));
    if  (semaphore_ptr == 0)
    {
        RAW_ASSERT(0);
    }

    //这是 raw-os 的 API
    raw_semaphore_create(semaphore_ptr, (RAW_U8 *)"name_ptr", count);

    sem->sem = semaphore_ptr;

    return ERR_OK;
}

void sys_sem_free(sys_sem_t *sem)
```

```
{
    if ((sem == NULL) || (sem->sem == NULL))
    {
        RAW_ASSERT(0);
    }

    raw_semaphore_delete(sem->sem);     //这是 raw-os 的 API

    raw_memset(sem->sem, sizeof( RAW_SEMAPHORE), 0);
    port_free(sem->sem);

    sem->sem = NULL;
}
```

还有几个函数就不一一列举了，如有疑问看 doc/sys_arch.txt

3)修改网卡框架文件：

netif/ethernetif.c

该文件是作者提供的网卡驱动和 lwip 的接口框架。

该文件中要改动的函数只有 3 个:

```
static void   low_level_init(struct netif *netif);
static err_t  low_level_output(struct netif *netif, struct pbuf *p);
static struct pbuf *low_level_input(struct netif *netif);

/* 你可以给网卡起个名字 */
/* Define those to better describe your network interface. */
#define IFNAME0 'e'
#define IFNAME1 '0'

/**
 * Helper struct to hold private data used to operate your ethernet
 * interface.
 * Keeping the ethernet address of the MAC in this struct is not
 * necessary as it is already kept in the struct netif.
 * But this is only an example, anyway...
 */
struct ethernetif
{
    struct eth_addr *ethaddr;
    // Add whatever per-interface state that is needed here.
    // 在这里添加网卡的私有数据，比如和网卡相关的信号量，互斥锁，
    // 网卡状态等等，这不是必须的
};
```

3 个网卡相关的函数只要改动红色部分，需根据具体的网卡驱动函数改动

```
static void low_level_init(struct netif *netif)
{
    struct ethernetif *ethernetif = netif->state;

    /* set MAC hardware address length */
    netif->hwaddr_len = ETHARP_HWADDR_LEN;

    /* 设置 MAC 地址，必须与网卡初始化的地址相同 */
    netif->hwaddr[0] = ;
    netif->hwaddr[1] = ;
    netif->hwaddr[2] = ;
    netif->hwaddr[3] = ;
    netif->hwaddr[4] = ;
    netif->hwaddr[5] = ;

    /* maximum transfer unit */
    netif->mtu = 1500;

    /* device capabilities */
    /* don't set NETIF_FLAG_ETHARP if this device is not an ethernet one */
    netif->flags = NETIF_FLAG_BROADCAST | NETIF_FLAG_ETHARP | NETIF_FLAG_LINK_UP;

    /* 在这里添加其他初始化代码(如真正的网卡初始化，phy 初始化等) */

}
```

```
static err_t low_level_output(struct netif *netif, struct pbuf *p)
{
    struct ethernetif *ethernetif = netif->state;
    struct pbuf *q;

    initiate transfer();

#if ETH_PAD_SIZE
    pbuf_header(p, -ETH_PAD_SIZE); /* drop the padding word */
#endif

    for(q = p; q != NULL; q = q->next) {
        /* Send the data from the pbuf to the interface, one pbuf at a
           time. The size of the data in each pbuf is kept in the ->len
           variable. */
        send data from(q->payload, q->len);
    }

  signal that packet should be sent();

#if ETH_PAD_SIZE
  pbuf_header(p, ETH_PAD_SIZE); /* reclaim the padding word */
#endif

  LINK_STATS_INC(link.xmit);

  return ERR_OK;
}
```

```
static struct pbuf * low_level_input(struct netif *netif)
{
    struct ethernetif *ethernetif = netif->state;
    struct pbuf *p, *q;
    u16_t len;

    /* Obtain the size of the packet and put it into the "len" variable. */
    len = ;     // 获取将要接收的数据长度

#if ETH_PAD_SIZE
    len += ETH_PAD_SIZE; /* allow room for Ethernet padding */
#endif

    /* We allocate a pbuf chain of pbufs from the pool. */
    p = pbuf_alloc(PBUF_RAW, len, PBUF_POOL);

    if (p != NULL) {

#if ETH_PAD_SIZE
        pbuf_header(p, -ETH_PAD_SIZE); /* drop the padding word */
#endif

        /* We iterate over the pbuf chain until we have read the entire
         * packet into the pbuf. */
        for(q = p; q != NULL; q = q->next) {
            /* Read enough bytes to fill this pbuf in the chain. The
             * available data in the pbuf is given by the q->len
             * variable.
             * This does not necessarily have to be a memcpy, you can also
             * preallocate pbufs for a DMA-enabled MAC and after receiving truncate
             * it to the actually received size. In this case, ensure the tot_len
             * member of the pbuf is the sum of the chained pbuf len members.
             */
            read data into(q->payload, q->len);
        }
        acknowledge that packet has been read();

#if ETH_PAD_SIZE
        pbuf_header(p, ETH_PAD_SIZE); /* reclaim the padding word */
#endif

        LINK_STATS_INC(link.recv);
    } else {
```

```
        drop packet();
        LINK_STATS_INC(link.memerr);
        LINK_STATS_INC(link.drop);
    }

    return p;
}
```

LwIP 的使用

LwIP 的初始化：

LwIP 的初始化必须在 RTOS 启动之后才可以进行，因为它的初始化代码使用了一些 OS 提供的功能!!!

初始化代码示例：

```
extern err_t ethernetif_init(struct netif *netif);
struct netif lpc1788_netif;
ip_addr_t e0ip, e0mask, e0gw;

/* tcpip_init 使用的回调函数，用于判断 tcpip_init 初始化完成 */
static void tcpip_init_done(void *pdat)
{
    *(int *)pdat = 0;
}

void  ethernetif_input(struct netif *netif);
// 一直调用 ethernetif_input 函数，从网卡读取数据
static void lwip_read_task(void *netif)
{
    while(1)
    {
        ethernetif_input(netif);
    }
}

void init_lwip()
{
    struct netif *pnetif = NULL;
    int flag = 1;

    tcpip_init(tcpip_init_done, &flag);      // lwip 协议栈的初始化
    while(flag);

    IP4_ADDR(&e0ip, 192,168,6,188);          // 设置网卡 ip
    IP4_ADDR(&e0mask, 255,255,255,0);        // 设置子网掩码
    IP4_ADDR(&e0gw, 192,168,6,1);            // 设置网关

    //给 lwip 添加网卡
    pnetif = netif_add(&lpc1788_netif, &e0ip, &e0mask, &e0gw,
                       NULL, ethernetif_init, tcpip_input);
    netif_set_default(pnetif);       // 设置该网卡为默认网卡
```

```
    netif_set_up(&lpc1788_netif);   // 启动网卡，可以唤醒 DHCP 等服务

    // 创建一个任务。这个任务负责不停的调用 ethernetif_input 函数从网卡读取数据
    raw_task_create(&lwip_read_obj, (RAW_U8  *)"lwip_read", &lpc1788_netif,
                            CONFIG_RAW_PRIO_MAX - 25, 0,  lwip_read_stk,
                            LWIP_READ_STK_SIZE ,  lwip_read_task, 1);
}
```