Day Day Up 笔记之 uCOS-II+LwIP 在 STM32F107 上移植

声明

XX 如是说——尊重他人劳动成果，也是一种美德……
转载请注明出处

闲话稍说

这篇笔记旨在讲解 LwIP 在 uCOS-II 的上的过程，着重讲述移植的整个流程，不拘泥于细节。从头至尾一步一步的完成移植工作。本篇目标是能够 ping 通目标板，PC 机与目标机建立一个初步的连接，至于说其他的应用服务，暂时不予讲述。

友情提示：要想顺利完成 LwIP 移植，你需具备：

1、有一定的 C 语言基础和数据结构的知识；
2、熟悉 uCOS-II，了解信号量，消息邮箱等通信方式；
3、知道 LwIP 是何物，能用来干什么

硬件平台

STM32 硬件平台　某开发板或你自己设计的硬件
　　我使用是自己的 STM32F107VC+DP83848 板子

IDE 环境

EWARM-IAR　V5.4

uCos-II　V2.86

LwIP　V1.3.2

目录

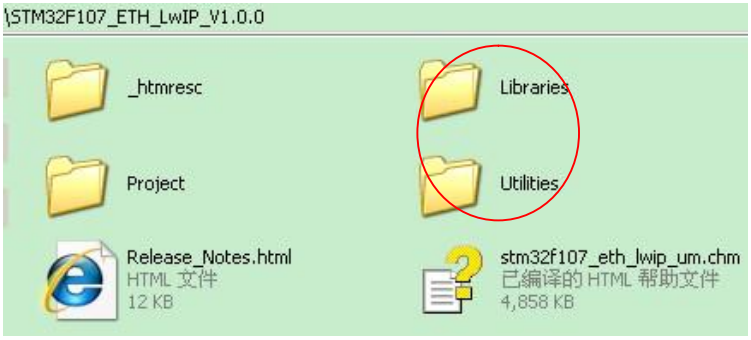
直奔主题

1、巧妇备米

1.1、下载 STM32F107_ETH_LwIP 例程

下载地址：
http://www.st.com/stonline/stappl/resourceSelector/app?page=resourceSelector&doctype=FIRMWARE&SubClassID=1169

这是一个“裸奔”的例子，解压后打开：



这里有一些文件和源码是我们需要的，比如说，STM32 网卡驱动，LwIP 源文件。后面会具体提到

1.2、基础工程的建立

这里直接使用之前已经建好且可用的一个 uCOS-II 工程，当然你可以自己从新建立一个或者用之前建立好了的。如果不知道怎么建立 uCOS-II，请详见《Day Day Up 之一步一步移植 uCOS-II 到 STM32 上——EWARM 篇》，此处不再赘述。

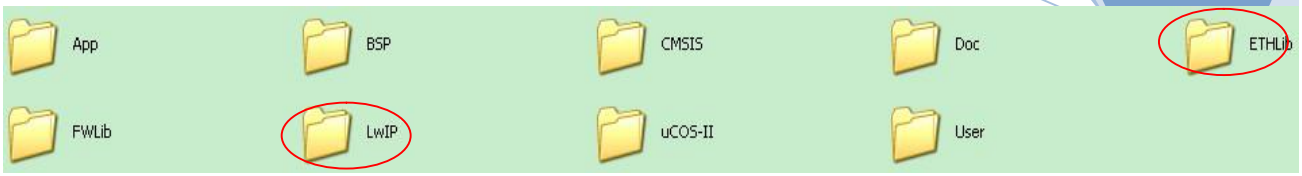
2、文件结构

先看我们文件结构的组织方式，注意:绿色标注部分

```
Project
|
|——App //应用软件任务
|       app.h
|       app.c
|       app_cfg.h
|       os_cfg.h
|——BSP //板级支持 硬件驱动程序
```

```
|       BSP.h
|       BSP.c
|——CMSIS //Cotex-M3 内核文件
|       core-m3.c
|       system_stm32f10x.c
|——DOC//说明等文档
|——ETHLib//stm32 网卡驱动库
|       stm32_eth.c
|——FWLib //STM32 官方库文件
|——LwIP
|       |——port
|       |
|       |——src
|       |       |——api
|       |       |——core
|       |       |——netif
|——Startup//启动文件
|——u/COS-II
|       |——Source //与硬件无关代码
|       |       os_core.c
|       |       os_flag.c
|       |       os_mbox.c
|       |       os_mem.c
|       |       os_mutex.c
|       |       os_q.c
|       |       os_sem.c
|       |       os_task.c
|       |       os_time.c
|       |       os_tmr.c
|       |       ucos-ii.h
|       |——Ports //与硬件相关
|       |       os_cpu.h
|       |       os_cpu_a.asm
|       |       os_cpu_c.c
|       |       os_dbg.c
|——User//工程建立在这个目录下
|       main.c
|       includes.h //主头文件
|       stm32f10x_it.c
```

A、按文件结构组织新增 ETHLib、LwIP 两文件夹

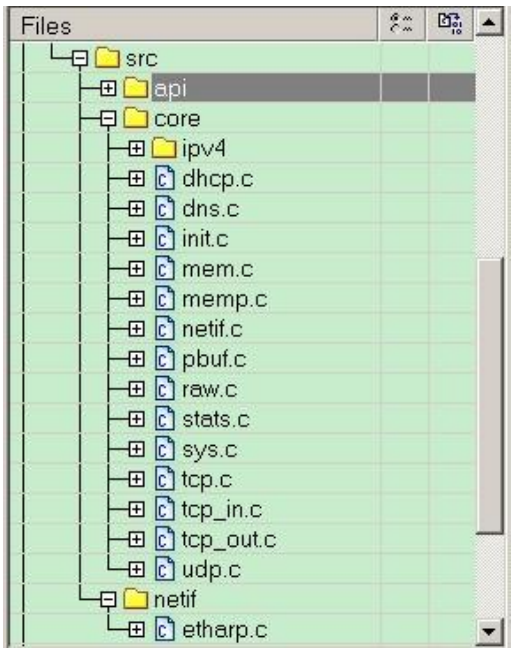


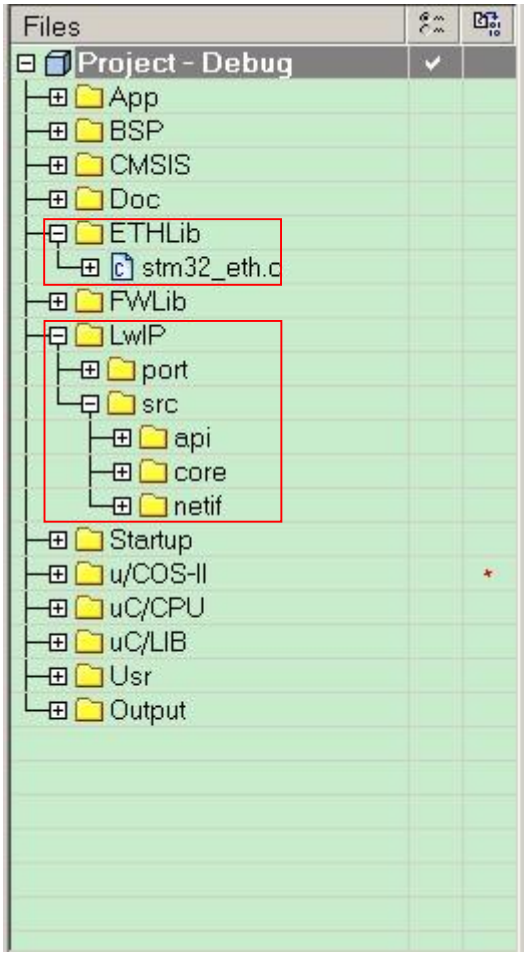
B、将 STM32F107_ETH_LwIP_V1.0.0\Libraries\STM32_ETH_Driver 的 inc 和 src 一并拷贝到 A 步骤的 ETHLib 文件夹下

C、拷贝 LwIP\src 源文件。将\STM32F107_ETH_LwIP_V1.0.0\Utilities\lwip-1.3.1\src 拷贝到 A 步骤的 LwIP 下面，为了移植的方便，将虽然不必要的\doc 和那几个说明文档一并拷贝[除\prot 外]，另外再 LwIP 目录下建立一个名为 port 的文件夹，后面的模拟层相关的代码文件会放到这里的

D、在 uCOS-II 基础工程下添加 ETHLib 组，添加相应的 c 文件，LwIP 组，src 组，再分别添加 api、core 和 netif 组到 src 组下，最后在 core 下面添加一个 ipv4 组。

特别说明：因为大多数网络是 ipv4，而 ppp、snmp 暂时不在考虑范围之内，所以 LwIP 下的 ipv6、ppp 和 snmp 就不添加进工程。另外，在 netif 下只加入 etharp.c

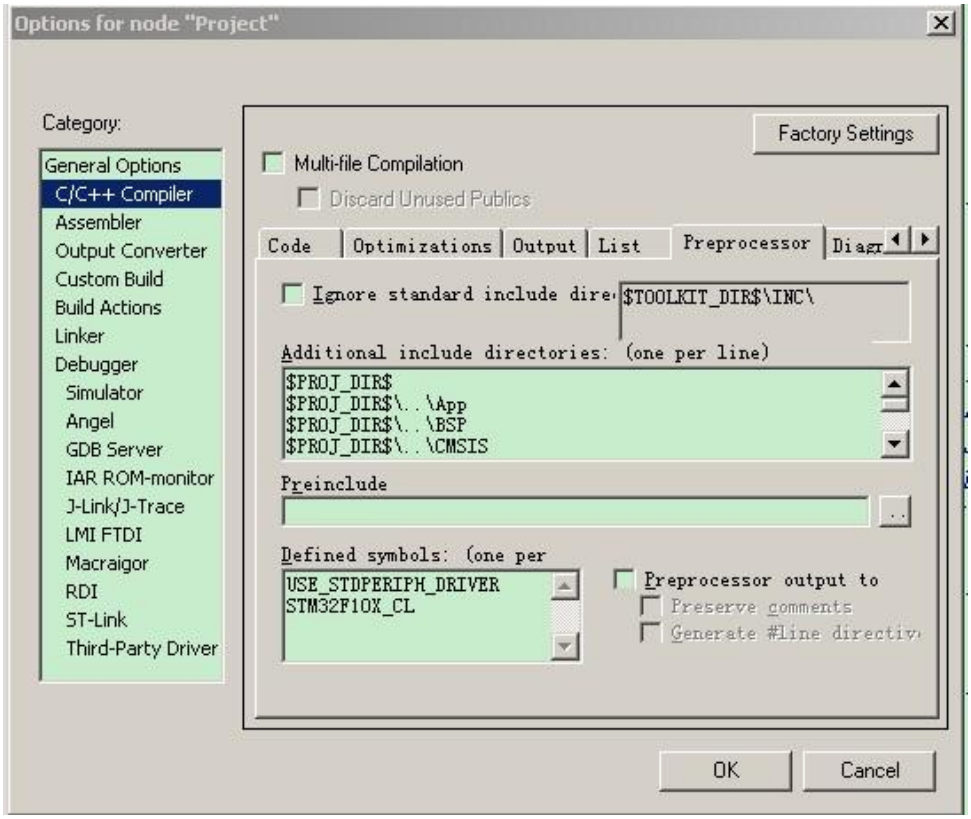




3、添加编译文件路径

设置新增文件的路径：

$PROJ_DIR$\..\ETHLib\inc
$PROJ_DIR$\..\ETHLib\src
$PROJ_DIR$\..\LwIP\src\include
$PROJ_DIR$\..\LwIP\src\include\lwip
$PROJ_DIR$\..\LwIP\src\include\ipv4
$PROJ_DIR$\..\LwIP\port



4、添加网口底层驱动代码

4.1、在 BSP.c 中添加：

```
/**
  * @brief  Ethernet Initialize function
  * @param  None
  * @retval None
  */
static void BSP_EthernetInit(void)
{
    /* Enable ETHERNET clock  */
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_ETH_MAC |
RCC_AHBPeriph_ETH_MAC_Tx |
                          RCC_AHBPeriph_ETH_MAC_Rx, ENABLE);

    /* Enable GPIOs clocks */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB
| RCC_APB2Periph_GPIOC |
                           RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE
| RCC_APB2Periph_AFIO
```

```
                                        , ENABLE);

  /* GPIO Config */
      GPIO_InitTypeDef GPIO_InitStructure;

      /* ETHERNET pins configuration */
      /* AF Output Push Pull:
      - ETH_MII_MDIO / ETH_RMII_MDIO: PA2
      - ETH_MII_MDC / ETH_RMII_MDC: PC1
      - ETH_MII_TXD2: PC2
      - ETH_MII_TX_EN / ETH_RMII_TX_EN: PB11
      - ETH_MII_TXD0 / ETH_RMII_TXD0: PB12
      - ETH_MII_TXD1 / ETH_RMII_TXD1: PB13
      - ETH_MII_PPS_OUT / ETH_RMII_PPS_OUT: PB5
      - ETH_MII_TXD3: PB8 */

      /* Configure PA2 as alternate function push-pull */
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
      GPIO_Init(GPIOA, &GPIO_InitStructure);

      /* Configure PC1 as alternate function push-pull */
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
      GPIO_Init(GPIOC, &GPIO_InitStructure);

      /* Configure PB5, PB11, PB12 and PB13 as alternate function push-pull */
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_11 |
                                    GPIO_Pin_12 | GPIO_Pin_13;
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
      GPIO_Init(GPIOB, &GPIO_InitStructure);

      /* Input (Reset Value):
      - ETH_MII_CRS CRS: PA0
      - ETH_MII_RX_CLK / ETH_RMII_REF_CLK: PA1
      - ETH_MII_COL: PA3
      - ETH_MII_RX_DV / ETH_RMII_CRS_DV: PA7
      - ETH_MII_TX_CLK: PC3
      - ETH_MII_RXD0 / ETH_RMII_RXD0: PC4
      - ETH_MII_RXD1 / ETH_RMII_RXD1: PC5
      - ETH_MII_RXD2: PB0
```

```
        - ETH_MII_RXD3: PB1
        - ETH_MII_RX_ER: PB10 */

        /* Configure PA1 and PA7 as input */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_7;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIOA, &GPIO_InitStructure);

        /* Configure PC4 and PC5 as input */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIOC, &GPIO_InitStructure); /**/

        /* MCO pin configuration------------------------------------------------ */
        /* Configure MCO (PA8) as alternate function push-pull */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_Init(GPIOA, &GPIO_InitStructure);

        Ethernet_Configuration();
}


/**
  * @brief  Configures the Ethernet Interface
  * @param  None
  * @retval None
  */
static void Ethernet_Configuration(void)
{
  ETH_InitTypeDef ETH_InitStructure;

  /* MII/RMII Media interface selection ----------------------------------------*/
#ifdef MII_MODE /* Mode MII with STM3210C-EVAL  */
  GPIO_ETH_MediaInterfaceConfig(GPIO_ETH_MediaInterface_MII);

  /* Get HSE clock = 25MHz on PA8 pin (MCO) */
  RCC_MCOConfig(RCC_MCO_HSE);

#elif defined RMII_MODE  /* Mode RMII with STM3210C-EVAL */
  GPIO_ETH_MediaInterfaceConfig(GPIO_ETH_MediaInterface_RMII);
```

```
  /* Set PLL3 clock output to 50MHz (25MHz /5 *10 =50MHz) */
  RCC_PLL3Config(RCC_PLL3Mul_10);
  /* Enable PLL3 */
  RCC_PLL3Cmd(ENABLE);
  /* Wait till PLL3 is ready */
  while (RCC_GetFlagStatus(RCC_FLAG_PLL3RDY) == RESET)
  {}

  /* Get PLL3 clock on PA8 pin (MCO) */
  RCC_MCOConfig(RCC_MCO_PLL3CLK);
#endif

  /* Reset ETHERNET on AHB Bus */
  ETH_DeInit();

  /* Software reset */
  ETH_SoftwareReset();

  /* Wait for software reset */
  while (ETH_GetSoftwareResetStatus() == SET);

  /* ETHERNET Configuration ------------------------------------------------------*/
  /* Call ETH_StructInit if you don't like to configure all ETH_InitStructure parameter
*/
  ETH_StructInit(&ETH_InitStructure);

  /* Fill ETH_InitStructure parametrs */
  /*-----------------------   MAC   -----------------------------------*/
  ETH_InitStructure.ETH_AutoNegotiation = ETH_AutoNegotiation_Enable   ;
  ETH_InitStructure.ETH_LoopbackMode = ETH_LoopbackMode_Disable;
  ETH_InitStructure.ETH_RetryTransmission = ETH_RetryTransmission_Disable;
  ETH_InitStructure.ETH_AutomaticPadCRCStrip =
ETH_AutomaticPadCRCStrip_Disable;
  ETH_InitStructure.ETH_ReceiveAll = ETH_ReceiveAll_Disable;
  ETH_InitStructure.ETH_BroadcastFramesReception =
ETH_BroadcastFramesReception_Enable;
  ETH_InitStructure.ETH_PromiscuousMode = ETH_PromiscuousMode_Disable;
  ETH_InitStructure.ETH_MulticastFramesFilter =
ETH_MulticastFramesFilter_Perfect;
  ETH_InitStructure.ETH_UnicastFramesFilter = ETH_UnicastFramesFilter_Perfect;
#ifdef CHECKSUM_BY_HARDWARE
  ETH_InitStructure.ETH_ChecksumOffload = ETH_ChecksumOffload_Enable;
#endif
```

```
  /*-----------------------      DMA      ----------------------------------*/

  /* When we use the Checksum offload feature, we need to enable the Store and
Forward mode:
  the store and forward guarantee that a whole frame is stored in the FIFO, so the
MAC can insert/verify the checksum,
  if the checksum is OK the DMA can handle the frame otherwise the frame is
dropped */
  ETH_InitStructure.ETH_DropTCPIPChecksumErrorFrame =
ETH_DropTCPIPChecksumErrorFrame_Enable;
  ETH_InitStructure.ETH_ReceiveStoreForward = ETH_ReceiveStoreForward_Enable;
  ETH_InitStructure.ETH_TransmitStoreForward =
ETH_TransmitStoreForward_Enable;

  ETH_InitStructure.ETH_ForwardErrorFrames = ETH_ForwardErrorFrames_Disable;
  ETH_InitStructure.ETH_ForwardUndersizedGoodFrames =
ETH_ForwardUndersizedGoodFrames_Disable;
  ETH_InitStructure.ETH_SecondFrameOperate =
ETH_SecondFrameOperate_Enable;
  ETH_InitStructure.ETH_AddressAlignedBeats = ETH_AddressAlignedBeats_Enable;
  ETH_InitStructure.ETH_FixedBurst = ETH_FixedBurst_Enable;
  ETH_InitStructure.ETH_RxDMABurstLength = ETH_RxDMABurstLength_32Beat;
  ETH_InitStructure.ETH_TxDMABurstLength = ETH_TxDMABurstLength_32Beat;
  ETH_InitStructure.ETH_DMAArbitration =
ETH_DMAArbitration_RoundRobin_RxTx_2_1;

  /* Configure Ethernet */
  ETH_Init(&ETH_InitStructure, PHY_ADDRESS);

  /* Enable the Ethernet Rx Interrupt */
  ETH_DMAITConfig(ETH_DMA_IT_NIS | ETH_DMA_IT_R, ENABLE);

}
```

4.2、对应的函数原型声明：

```
static void BSP_EthernetInit(void);
static void Ethernet_Configuration(void);
```

4.3、在 BSP_Init()中调用 BSP_EthernetInit()

……

```
BSP_EthernetInit();
……
```

4.3、相关宏定义：

```
#define DP83848_PHY             /* Ethernet pins mapped on STM32F107 Board */
#define PHY_ADDRESS         0x01 /* Relative to STM32F107 Board */
//#define MII_MODE
#define RMII_MODE               // STM32F107 connect PHY using RMII mode
```

4.4、包含 stm32_eth.h 头文件：

```
#include "stm32_eth.h"
```

5、编写操作系统模拟层代码

这将是我们本篇的重点，也是移植 LwIP 的关键内容。在本篇一开头的 1.1 提到过一个 sys_arch.txt 文件，事实上，这个文档较为详细的说明了操作系统模拟层的移植，也是 LwIP 作者提供的为数不多的文档，要珍惜。

5.1、sys_arch.txt 的中文翻译

LwIP 0.6++ sys_arch 接口
作者：Adam Dunkels

操作系统模拟层（sys_arch）存在的目的主要是为了方便 LwIP 的移植，它在底层操作系统和 LwIP 之间提供了一个接口。这样，我们在移植 LwIP 到一个新的目标系统时，只需修改这个接口即可。不过，不依赖底层操作系统的支持也可以实现这个接口。

sys_arch 需要为 LwIP 提供信号量（semaphores）和邮箱（mailboxes）两种进程间通讯方式（IPC）。如果想获得 LwIP 的完整功能，sys_arch 还必须支持多线程。当然，对于仅需要基本功能的用户来说，可以不去实现多线程。LwIP 以前的版本还要求 sys_arch 实现定时器调度，不过，从 LwIP0.5 开始，这一需求在更高一层实现。除了上文所述的 sys_arch 源文件需要实现的功能外，LwIP 还要求用户提供几个头文件，这几个头文件包含 LwIP 使用的宏定义。下文将详细讲述 sys_arch 及头文件的实现。

信号量即可以是计数信号量，也可以是二值信号量——LwIP 都可以正常工作。邮箱用于消息传递，用户即可以将其实现为一个队列，允许多条消息投递到这个邮箱，也可以每次只允许投递一个消息。这两种方式 LwIP 都可以正常运作。不过，前者更加有效。需要用户特别注意的是——投递到邮箱中的消息只能是一

个指针。

在 sys_arch.h 文件中，我们指定数据类型“sys_sem_t”表示信号量，“sys_mbox_t”表示邮箱。至于 sys_sem_t 和 sys_mbox_t 如何表示这两种不同类型，LwIP 没有任何限制。

以下函数必须在 sys_arch 中实现：

- void sys_init(void)
初始化 sys_arch 层。

- sys_sem_t sys_sem_new(u8_t count)
建立并返回一个新的信号量。参数 count 指定信号量的初始状态。

- void sys_sem_free(sys_sem_t sem)
释放信号量。

- void sys_sem_signal(sys_sem_t sem)
发送一个信号。

- u32_t sys_arch_sem_wait(sys_sem_t sem, u32_t timeout)
等待指定的信号并阻塞线程。timeout 参数为 0，线程会一直被阻塞至收到指定的信号；非 0，则线程仅被阻塞至指定的 timeout 时间（单位为毫秒）。在 timeout 参数值非 0 的情况下，返回值为等待指定的信号所消耗的毫秒数。如果在指定的时间内并没有收到信号，返回值为 SYS_ARCH_TIMEOUT。如果线程不必再等待这个信号（也就是说，已经收到信号），返回值也可以为 0。注意，LwIP 实现了一个名称与之相似的函数来调用这个函数，sys_sem_wait()，注意区别。

- sys_mbox_t sys_mbox_new(void)
建立一个空的邮箱。

- void sys_mbox_free(sys_mbox_t mbox)
释放一个邮箱。如果释放时邮箱中还有消息，它表明 LwIP 中存在一个编程错误，应该通知开发者（原文如此，这句话很费解。个人理解的意思是：当执行 sys_mbox_free()这个函数时，按道理邮箱中不应该再存在任何消息，如果用户使用 LwIP 时发现邮箱中还存在消息，说明 LwIP 的开发者存在一个编程错误，不能把邮箱中的消息全部取出并处理掉。遇到这种情况，用户应该告诉 LwIP 的作者，纠正这个 bug，译注）。

- void sys_mbox_post(sys_mbox_t mbox, void *msg)
投递消息“msg”到指定的邮箱“mbox”。

- u32_t sys_arch_mbox_fetch(sys_mbox_t mbox, void **msg, u32_t timeout)
阻塞线程直至邮箱收到至少一条消息。最长阻塞时间由 timeout 参数指定（与 sys_arch_sem_wait()函数类似）。msg 是一个结果参数，用来保存邮箱中的消息指针（即*msg = ptr），它的值由这个函数设置。“msg”参数有可能为空，这表明当前这条消息应该被丢弃。 返回值与 sys_arch_sem_wait()函数相同：等待的毫秒数或者 SYS_ARCH_TIMEOUT——如果时间溢出的话。LwIP 实现的函数中，

有一个名称与之相似的——sys_mbox_fetch()，注意区分。

- struct sys_timeouts *sys_arch_timeouts(void)

返回一个指向当前线程使用的 sys_timeouts 结构的指针。LwIP 中，每一个线程都有一个 timeouts 链表，这个链表由 sys_timeout 结构组成，sys_timeouts 结构则保存了指向这个链表的指针。这个函数由 LwIP 的超时调度程序调用，并且不能返回一个空（NULL）值。 单线程 sys_arch 实现中，这个函数只需简单返回一个指针即可。这个指针指向保存在 sys_arch 模块中的 sys_timeouts 全局变量。

如果底层操作系统支持多线程并且 LwIP 中需要这样的功能，那么，下面的函数必须实现：

- sys_thread_t sys_thread_new(void(*thread)(void *arg), void *arg, int prio)

启动一个由函数指针 thread 指定的新线程，arg 将作为参数传递给 thread()函数，prio 指定这个新线程的优先级。返回值为这个新线程的 ID，ID 和优先级由底层操作系统决定。

- sys_prot_t sys_arch_protect(void)

这是一个可选函数，它负责完成临界区域保护并返回先前的保护状态。该函数只有在小的临界区域需要保护时才会被调用。基于 ISR 驱动的嵌入式系统可以通过禁止中断来实现这个函数。基于任务的系统可以通过互斥量或禁止任务来实现这个函数。该函数应该支持来自于同一个任务或中断的递归调用。换句话说，当该区域已经被保护，sys_arch_protect()函数依然能被调用。这时，函数的返回值会通知调用者该区域已经被保护。
如果你的移植正在支持一个操作系统，sys_arch_protect()函数仅仅是一个需要。

- void sys_arch_unprotect(sys_prot_t pval)

该函数同样是一个可选函数。它的功能就是恢复受保护区域的先前保护状态，先前是受到保护还是没有受到保护由参数 pval 指定。它与 sys_arch_protect()函数配套使用，详细信息参看 sys_arch_protect()函数。
该函数的说明是按照译者个人理解的意思翻译，原文讲述不是很清楚，如有错误，欢迎批评指正，译注。

OS 支持的模拟层需要添加的头文件说明

- cc.h　　与硬件平台及编译器相关的环境变量及数据类型声明文件（一些或许应该移到 sys_arch.h 文件）。

LwIP 使用的数据类型定义——u8_t, s8_t, u16_t，s16_t，u32_t，s32_t，mem_ptr_t。

与编译器相关的 LwIP 结构体封装宏：

PACK_STRUCT_FIELD(x)
PACK_STRUCT_STRUCT

PACK_STRUCT_BEGIN
PACK_STRUCT_END

与平台相关的调试输出：
LWIP_PLATFORM_DIAG(X)　- 非故障，输出一条提示信息。
LWIP_PLATFORM_ASSERT(x) - 故障，输出一条故障信息并放弃执行。

“轻便的（lightweight）”同步机制：
SYS_ARCH_DECL_PROTECT(x)　- 声明一个保护状态变量。
SYS_ARCH_PROTECT(x)　- 进入保护模式。
SYS_ARCH_UNPROTECT(x)　- 脱离保护模式。

如果编译器不提供 memset()函数，这个文件必须包含它的定义，或者包含（include）一个定义它的文件。
这个文件要么包含一个本地系统（system-local）提供的头文件<errno.h>——这个文件定义了标准的*nix 错误编码，要么增加一条宏定义语句：#define LWIP_PROVIDE_ERRNO，这将使得 lwip/arch.h 头文件来定义这些编码。这些编码被用于 LwIP 的各个部分。

- perf.h　定义了性能测量使用的宏，由 LwIP 调用，可以将其定义为一个空的宏。
PERF_START　- 开始测量。
PERF_STOP(x)　- 结束测量并记录结果。

- sys_arch.h　sys_arch.c 的头文件。
定义 Arch（即整个移植所依赖的操作系统平台，译注）需要的数据类型：sys_sem_t，sys_mbox_t，sys_thread_t，以及可选类型：sys_prot_t。

sys_mbox_t 和 sys_sem_t 变量的 NULL 值定义：
SYS_MBOX_NULL　NULL
SYS_SEM_NULL　NULL

上面这段内容剽窃自《uC/OS-II 平台下的 LwIP 移植笔记》——作者：焦海波

5.2、编写模拟层代码

A、在\LwIP\port 下新建 sys_arch.c 文件，添加到工程的 LwIP\port 下

B、在\LwIP\port 下新建 arch 文件夹，新建一个 sys_arch 的.h 头文件，还有 cc.h

C、在 cc.h 下定义常用数据类型，服务于模拟层接口函数和底层协议。添加如下代码：

```
#ifndef __CC_H__
#define __CC_H__

#include "cpu.h"
#include "stdio.h"

/*-------------data type----------------------------------------------------*/

typedef unsigned     char       u8_t;      /* Unsigned 8 bit quantity          */
typedef signed       char       s8_t;      /* Signed       8 bit quantity      */
typedef unsigned     short      u16_t;     /* Unsigned 16 bit quantity         */
typedef signed       short      s16_t;     /* Signed      16 bit quantity      */
typedef unsigned     long       u32_t;     /* Unsigned 32 bit quantity         */
typedef signed       long       s32_t;     /* Signed      32 bit quantity      */
typedef u32_t mem_ptr_t;                   /* Unsigned 32 bit quantity         */
//typedef int sys_prot_t;

/*-------------critical region protection (depends on uC/OS-II setting)-------*/

#if OS_CRITICAL_METHOD == 1
#define SYS_ARCH_DECL_PROTECT(lev)
#define SYS_ARCH_PROTECT(lev)        CPU_INT_DIS()
#define SYS_ARCH_UNPROTECT(lev)          CPU_INT_EN()
#endif

#if OS_CRITICAL_METHOD == 3    //method 3 is used in this port.
#define SYS_ARCH_DECL_PROTECT(lev)      u32_t lev
#define SYS_ARCH_PROTECT(lev)        lev = OS_CPU_SR_Save()
#define SYS_ARCH_UNPROTECT(lev)          OS_CPU_SR_Restore(lev)
```

```
#endif

/*-----------------------------------------------------------------------------*/

/* define compiler specific symbols */
#if defined (__ICCARM__)

#define PACK_STRUCT_BEGIN
#define PACK_STRUCT_STRUCT
#define PACK_STRUCT_END
#define PACK_STRUCT_FIELD(x) x
#define PACK_STRUCT_USE_INCLUDES

#elif defined (__CC_ARM)

#define PACK_STRUCT_BEGIN __packed
#define PACK_STRUCT_STRUCT
#define PACK_STRUCT_END
#define PACK_STRUCT_FIELD(x) x

#elif defined (__GNUC__)

#define PACK_STRUCT_BEGIN
#define PACK_STRUCT_STRUCT __attribute__ ((__packed__))
#define PACK_STRUCT_END
#define PACK_STRUCT_FIELD(x) x

#elif defined (__TASKING__)

#define PACK_STRUCT_BEGIN
#define PACK_STRUCT_STRUCT
#define PACK_STRUCT_END
#define PACK_STRUCT_FIELD(x) x

#endif

/*---define (sn)printf formatters for these lwip types, for lwip DEBUG/STATS--*/

#define U16_F "4d"
#define S16_F "4d"
#define X16_F "4x"
#define U32_F "8ld"
#define S32_F "8ld"
#define X32_F "8lx"
```

```
/*--------------macros------------------------------------------------------*/
#ifndef LWIP_PLATFORM_ASSERT
#define LWIP_PLATFORM_ASSERT(x) \
      do \
      {     printf("Assertion \"%s\" failed at line %d in %s\n", x, __LINE__, __FILE__); \
      } while(0)
#endif

#ifndef LWIP_PLATFORM_DIAG
#define LWIP_PLATFORM_DIAG(x) do {printf x;} while(0)
#endif

#endif /* __CC_H__ */
```

D、留意到#include "cpu.h"没？这可不是 uC/OS-II 源码中的那个 cpu.h，

新建 cpu.h，保存到\LwIP\port\arch 目录下
添加如下代码

```
#ifndef __CPU_H__
#define __CPU_H__

#define BYTE_ORDER LITTLE_ENDIAN      //小端模式

#endif /* __CPU_H__ */
```

虽然只有一行代码，但是确实必须的

D、编写 sys_arch.h

并添加如下代码：

```
#ifndef __ARCH_SYS_ARCH_H__
#define __ARCH_SYS_ARCH_H__

#include "arch/cc.h" //包含 cc.h 头文件
#include "ucos_ii.h"

#ifdef SYS_ARCH_GLOBALS
#define SYS_ARCH_EXT
#else
#define SYS_ARCH_EXT extern
#endif

#define MAX_QUEUES              10      // the number of mailboxes
```

```
#define MAX_QUEUE_ENTRIES 20 // the max size of each mailbox

#define SYS_MBOX_NULL (void *)0
#define SYS_SEM_NULL    (void *)0

#define sys_arch_mbox_tryfetch(mbox,msg) \
        sys_arch_mbox_fetch(mbox,msg,1)

/*----------------type define----------------------------------------------*/

/** struct of LwIP mailbox */
typedef struct {
      OS_EVENT*      pQ;
      void*          pvQEntries[MAX_QUEUE_ENTRIES];
} TQ_DESCR, *PQ_DESCR;

typedef OS_EVENT *sys_sem_t; // type of semiphores
typedef PQ_DESCR sys_mbox_t; // type of mailboxes
typedef INT8U sys_thread_t; // type of id of the new thread

typedef INT8U sys_prot_t;

#endif /* __SYS_RTXC_H__ */
```

E、编写信号量操作函数

在 sys_arch.c 中

-sys_sem_new()

```
/*-----------------------------------------------------------------------------------*/
/*
   Creates and returns a new semaphore. The "count" argument specifies
   the initial state of the semaphore. TBD finish and test
*/
sys_sem_t
sys_sem_new(u8_t count)
{
     sys_sem_t pSem;

          pSem = OSSemCreate((u16_t)count);
          LWIP_ASSERT("OSSemCreate ",pSem != NULL );
          return pSem;
}
```

- sys_sem_signal()

```
/*-----------------------------------------------------------------------------------*/
/*
   Signals a semaphore
*/
void
sys_sem_signal(sys_sem_t sem)
{
      u8_t ucErr;
      ucErr = OSSemPost((OS_EVENT *)sem);

      // May be called when a connection is already reset, should not check...
      // LWIP_ASSERT( "OSSemPost ", ucErr == OS_NO_ERR );
}
```

-sys_sem_free()

```
/*-----------------------------------------------------------------------------------*/
/*
   Deallocates a semaphore
*/
void
sys_sem_free(sys_sem_t sem)
{
      u8_t        ucErr;
      (void)OSSemDel( (OS_EVENT *)sem, OS_DEL_ALWAYS, &ucErr );
      LWIP_ASSERT( "OSSemDel ", ucErr == OS_NO_ERR );
}
```

- sys_arch_sem_wait()

```
/*-----------------------------------------------------------------------------------*/
/*
   Blocks the thread while waiting for the semaphore to be
   signaled. If the "timeout" argument is non-zero, the thread should
   only be blocked for the specified time (measured in
   milliseconds).

   If the timeout argument is non-zero, the return value is the number of
   milliseconds spent waiting for the semaphore to be signaled. If the
   semaphore wasn't signaled within the specified time, the return value is
   SYS_ARCH_TIMEOUT. If the thread didn't have to wait for the semaphore
   (i.e., it was already signaled), the function may return zero.

   Notice that lwIP implements a function with a similar name,
   sys_sem_wait(), that uses the sys_arch_sem_wait() function.
```

```
*/
u32_t
sys_arch_sem_wait(sys_sem_t sem, u32_t timeout)
{
    u8_t ucErr;
    u32_t ucos_timeout, timeout_new;

    // timeout 单位以 ms 计转换为 ucos_timeout 单位以 TICK 计
    if(timeout != 0)
    {
        ucos_timeout = (timeout * OS_TICKS_PER_SEC) / 1000; // convert to
timetick
        if(ucos_timeout < 1)
            ucos_timeout = 1;
        else if(ucos_timeout > 65536) // uC/OS only support u16_t pend
            ucos_timeout = 65535; // 最多等待 TICK 数 这是 uC/OS 所能 处理
的最大延时
    }
    else ucos_timeout = 0;

    timeout = OSTimeGet(); // 记录起始时间

    OSSemPend ((OS_EVENT *)sem,(u16_t)ucos_timeout, (u8_t *)&ucErr);

    if(ucErr == OS_TIMEOUT)
        timeout = SYS_ARCH_TIMEOUT;  // only when timeout!
    else
    {
       //LWIP_ASSERT( "OSSemPend ", ucErr == OS_NO_ERR );
        //for pbuf_free, may be called from an ISR

        timeout_new = OSTimeGet(); // 记录终止时间
        if (timeout_new>=timeout) timeout_new = timeout_new - timeout;
        else timeout_new = 0xffffffff - timeout + timeout_new;

        timeout = (timeout_new * 1000 / OS_TICKS_PER_SEC + 1); //convert to
milisecond 为什么加 1?
    }

    return timeout;
}
```

F、编写邮箱操作函数

-sys_mbox_t()

```
/*-----------------------------------------------------------------------------------*/
/*
   Creates an empty mailbox.
*/
sys_mbox_t
sys_mbox_new(int size)
{
      // prarmeter "size" can be ignored in your implementation.
      u8_t            ucErr;
      PQ_DESCR        pQDesc;

      pQDesc = OSMemGet( pQueueMem, &ucErr );
      LWIP_ASSERT("OSMemGet ", ucErr == OS_NO_ERR );

      if( ucErr == OS_NO_ERR )
      {
            if( size > MAX_QUEUE_ENTRIES ) //  邮箱最多容纳 MAX_QUEUE_ENTRIES
消息数目
                  size = MAX_QUEUE_ENTRIES;

            pQDesc->pQ = OSQCreate( &(pQDesc->pvQEntries[0]), size );
            LWIP_ASSERT( "OSQCreate ", pQDesc->pQ != NULL );

            if( pQDesc->pQ != NULL )
                  return pQDesc;
            else
            {
                  ucErr = OSMemPut( pQueueMem, pQDesc );
                  return SYS_MBOX_NULL;
            }
      }
      else return SYS_MBOX_NULL;
}
```

- sys_mbox_free()

```
void
sys_mbox_free(sys_mbox_t mbox)
{
      u8_t        ucErr;
```

```
    LWIP_ASSERT( "sys_mbox_free ", mbox != SYS_MBOX_NULL );

    //clear OSQ EVENT
    OSQFlush( mbox->pQ );

    //del OSQ EVENT
    (void)OSQDel( mbox->pQ, OS_DEL_NO_PEND, &ucErr);
    LWIP_ASSERT( "OSQDel ", ucErr == OS_NO_ERR );

    //put mem back to mem queue
    ucErr = OSMemPut( pQueueMem, mbox );
    LWIP_ASSERT( "OSMemPut ", ucErr == OS_NO_ERR );
}
```

-sys_mbox_post()

```
/*-----------------------------------------------------------------------------------*/
/*
   Posts the "msg" to the mailbox.
*/
void
sys_mbox_post(sys_mbox_t mbox, void *msg)
{
    u8_t ubErr,i=0;

    if( msg == NULL ) msg = (void*)&pvNullPointer;

    while((i<10) && ((ubErr = OSQPost( mbox->pQ, msg)) != OS_NO_ERR))
    {
        i++;//if full, try 10 times
        OSTimeDly(5);
    }
    //if (i==10) printf("sys_mbox_post error!\n");
}
```

-sys_mbox_trypost()

```
/*-----------------------------------------------------------------------------------*/
/*
   Try to post the "msg" to the mailbox.
*/
err_t sys_mbox_trypost(sys_mbox_t mbox, void *msg)
{
    u8_t ubErr;

    if(msg == NULL ) msg = (void*)&pvNullPointer;
```

```
    if((ubErr = OSQPost( mbox->pQ, msg)) != OS_NO_ERR)
        return ERR_MEM;

    return ERR_OK;
}
```

- sys_arch_mbox_fetch()

```
/*-----------------------------------------------------------------------------------*/
/*
  Blocks the thread until a message arrives in the mailbox, but does
  not block the thread longer than "timeout" milliseconds (similar to
  the sys_arch_sem_wait() function). The "msg" argument is a result
  parameter that is set by the function (i.e., by doing "*msg =
  ptr"). The "msg" parameter maybe NULL to indicate that the message
  should be dropped.

  The return values are the same as for the sys_arch_sem_wait() function:
  Number of milliseconds spent waiting or SYS_ARCH_TIMEOUT if there was a
  timeout.

  Note that a function with a similar name, sys_mbox_fetch(), is
  implemented by lwIP.
*/
u32_t sys_arch_mbox_fetch(sys_mbox_t mbox, void **msg, u32_t timeout)
{
    u8_t    ucErr;
    u32_t   ucos_timeout, timeout_new;
    void    *temp;

    if(timeout != 0)
    {
        ucos_timeout = (timeout * OS_TICKS_PER_SEC)/1000; //convert to timetick

        if(ucos_timeout < 1)
            ucos_timeout = 1;
        else if(ucos_timeout > 65535) //ucOS only support u16_t timeout
            ucos_timeout = 65535;
    }
    else ucos_timeout = 0;

    timeout = OSTimeGet();

    temp = OSQPend( mbox->pQ, (u16_t)ucos_timeout, &ucErr );
```

```
    if(msg != NULL)
    {
        if( temp == (void*)&pvNullPointer )
                *msg = NULL;
        else
                *msg = temp;
    }

    if ( ucErr == OS_TIMEOUT )
          timeout = SYS_ARCH_TIMEOUT;
    else
    {
        LWIP_ASSERT( "OSQPend ", ucErr == OS_NO_ERR );

          timeout_new = OSTimeGet();
          if (timeout_new>timeout) timeout_new = timeout_new - timeout;
          else timeout_new = 0xffffffff - timeout + timeout_new;

          timeout = timeout_new * 1000 / OS_TICKS_PER_SEC + 1; //convert to
milisecond
    }

    return timeout;
}
```

G、初始化 sys_arch 层

-sys_init()

```
/*-----------------------------------------------------------------------------------*/
/*
   Initialize sys arch
*/
void
sys_init(void)
{

    u8_t i, ucErr;

        // init mem used by sys_mbox_t, use ucosII functions
        //  指定内存起始地址以 4 字节对齐
        pQueueMem = OSMemCreate(
```

```
(void*)((u32_t)((u32_t)pcQueueMemoryPool+MEM_ALIGNMENT-1)                    &
~(MEM_ALIGNMENT-1)),
                                                MAX_QUEUES, sizeof(TQ_DESCR), &ucErr
                                                    );

            LWIP_ASSERT( "OSMemCreate ", ucErr == OS_NO_ERR );

    // Initialize the the per-thread sys_timeouts structures
    // make sure there are no valid pids in the list
    for(i = 0; i < LWIP_TASK_MAX; i++)
    {
                lwip_timeouts[i].next = NULL;
    }
}
```

H、编写 sys_arch_timeouts 函数

```
/*-----------------------------------------------------------------------------------*/
/*
   Returns a pointer to the per-thread sys_timeouts structure. In lwIP,
   each thread has a list of timeouts which is represented as a linked
   list of sys_timeout structures. The sys_timeouts structure holds a
   pointer to a linked list of timeouts. This function is called by
   the lwIP timeout scheduler and must not return a NULL value.

   In a single threaded sys_arch implementation, this function will
   simply return a pointer to a global sys_timeouts variable stored in
   the sys_arch module.
*/
struct sys_timeouts *
sys_arch_timeouts(void)
{
      u8_t curr_prio;
      s8_t ubldx;

      OS_TCB curr_task_pcb;

      null_timeouts.next = NULL;

      OSTaskQuery(OS_PRIO_SELF,&curr_task_pcb);
      curr_prio = curr_task_pcb.OSTCBPrio;

      ubldx = (u8_t)(curr_prio - LWIP_START_PRIO);
```

```
    if((ubldx>=0) && (ubldx<LWIP_TASK_MAX))
    {
        //printf("\nlwip_timeouts[%d],prio=%d!!! \n",ubldx,curr_prio);
        return &lwip_timeouts[ubldx];
    }
    else
    {
        //printf("\nnull_timeouts,prio=%d!!! \n",curr_prio);
        return &null_timeouts;
    }
}
```

I、编写 thread_t sys_thread_new 函数

建立一个新线程

```
/*-----------------------------------------------------------------------------------*/
/*
  Starts a new thread with priority "prio" that will begin its execution in the
  function "thread()". The "arg" argument will be passed as an argument to the
  thread() function. The id of the new thread is returned. Both the id and
  the priority are system dependent.
*/
sys_thread_t sys_thread_new(char *name, void (* thread)(void *arg), void *arg, int
stacksize, int prio)
{
    u8_t ubPrio = 0;
    u8_t ucErr;

    arg = arg;

    if((prio > 0) && (prio <= LWIP_TASK_MAX))
    {
        ubPrio = (u8_t)(LWIP_START_PRIO + prio - 1);

        if(stacksize > LWIP_STK_SIZE)    // 任务堆栈大小不超过 LWIP_STK_SIZE
            stacksize = LWIP_STK_SIZE;

#if (OS_TASK_STAT_EN == 0)
        OSTaskCreate(thread,                (void                *)arg,
&LWIP_TASK_STK[prio-1][stacksize-1],ubPrio);
#else
        OSTaskCreateExt(thread,             (void                *)arg,
```

```
&LWIP_TASK_STK[prio-1][stacksize-1],ubPrio
                                    ,ubPrio,&LWIP_TASK_STK[prio-1][0],stacksize,(void
*)0,OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);
#endif
            OSTaskNameSet(ubPrio, (u8_t*)name, &ucErr);

//              return ubPrio;
      }
            return ubPrio;
}
```

在 sys_arch.h 中添加

```
#include "app_cfg.h" // define LWIP TASK Prio

/*-----------------macros--------------------------------------------------------*/
#define LWIP_STK_SIZE    300

#define LWIP_TASK_MAX    (LWIP_TASK_END_PRIO - LWIP_TASK_START_PRIO + 1)

//max number of lwip tasks (TCPIP) note LWIP TASK start with 1

#define LWIP_START_PRIO        LWIP_TASK_START_PRIO          //first prio of lwip
tasks in uC/OS-II

/*-----------------global variables----------------------------------------*/

SYS_ARCH_EXT OS_STK LWIP_TASK_STK[LWIP_TASK_MAX][LWIP_STK_SIZE];
```

在 uC/OS-II 配置文件 app_cfg.h 中添加：

```
#define    LWIP_TASK_START_PRIO                    10
#define    LWIP_TASK_END_PRIO                      12
```

K、添加 sys_arch.c 所需的头文件及变量定义等

```
#define SYS_ARCH_GLOBALS

/* lwIP includes. */
#include "lwip/debug.h"
#include "lwip/def.h"
#include "lwip/sys.h"
#include "lwip/mem.h"
```

```
#include "arch/sys_arch.h"

static OS_MEM *pQueueMem;

const void * const pvNullPointer = (mem_ptr_t*)0xffffffff;

static char pcQueueMemoryPool[MAX_QUEUES * sizeof(TQ_DESCR) + MEM_ALIGNMENT - 1];

//SYS_ARCH_EXT OS_STK LWIP_TASK_STK[LWIP_TASK_MAX][LWIP_STK_SIZE];

/* Message queue constants. */
#define archMESG_QUEUE_LENGTH    ( 6 )
#define archPOST_BLOCK_TIME_MS    ( ( unsigned portLONG ) 10000 )

struct sys_timeouts lwip_timeouts[LWIP_TASK_MAX];
struct sys_timeouts null_timeouts;
```

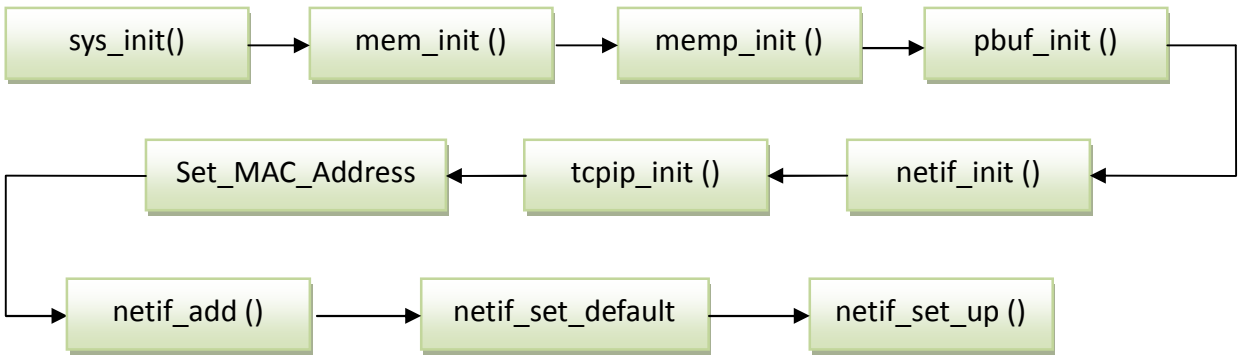
5.3、组织编写 LwIP 接口函数

我们需要定义一些接口函数与上层应用打交道，这一节就是为此服务的。主要功能是初始化 LwIP。这里我们可以参考 LwIP 源文件 doc 下面的 rawapi.txt 文档，它描述了 LwIP 初始化的一般流程，具体实现过程中，与描述的略有不同，要灵活变通。

A、新建两个文件，分别命名为:LwIP.c、LwIP.h。

[当然，也可以定义成其他名字。比如说，STM32F107_ETH_LwIP 例程中，就命名为：netconf]，保存到 LwIP\port，然后将 LwIP.c 添加到工程目录 LwIP 组下的 port 下

B、初始化流程：



C、函数的实现[参考 netconf.c]

在 LwIP.c

- Init_LwIP()

```
/**
* @brief Init_LwIP initialize the LwIP
*/
void Init_LwIP(void)
{
	struct ip_addr ipaddr;
	struct ip_addr netmask;
	struct ip_addr gw;
	uint8_t macaddress[6]={0,0,0,0,0,1};

	sys_sem_t sem;

	sys_init();

	/* Initializes the dynamic memory heap defined by MEM_SIZE.*/
	mem_init();

	/* Initializes the memory pools defined by MEMP_NUM_x.*/
	memp_init();

	pbuf_init();
	netif_init();

	printf("TCP/IP initializing... \r\n");
	sem = sys_sem_new(0);
	tcpip_init(TcpipInitDone, &sem);
	sys_sem_wait(sem);
	sys_sem_free(sem);
	printf("TCP/IP initialized. \r\n");

#if LWIP_DHCP
	/*  启用 DHCP 服务器  */
	ipaddr.addr = 0;
	netmask.addr = 0;
	gw.addr = 0;
#else
```

```
    /*  启用静态 IP */
  IP4_ADDR(&ipaddr, 192, 168, 1, 8);
  IP4_ADDR(&netmask, 255, 255, 255, 0);
  IP4_ADDR(&gw, 192, 168, 1, 1);
#endif

    Set_MAC_Address(macaddress);

    netif_add(&netif,  &ipaddr,  &netmask,  &gw,  NULL,  &ethernetif_init,
&tcpip_input);
    netif_set_default(&netif);

#if LWIP_DHCP
    dhcp_start(&netif);
#endif
    netif_set_up(&netif);
}
```

各个函数具体是怎么实现的，这里就不做说明了。注意 ethernetif_init，它将是下一节的内容。

-LwIP_Pkt_Handle()

```
/**
* @brief    Ethernet ISR
*/
void LwIP_Pkt_Handle(void)
{
  /* Read a received packet from the Ethernet buffers and send it to the lwIP for
handling */
  ethernetif_input(&netif);
}
```

D、其他相关代码

在 LwIP.c 中添加：

```
#define __LW_IP_C

/* Includes ------------------------------------------------------------------*/
#include "lwip/memp.h"
#include "LwIP.h"
#include "lwip/tcp.h"
#include "lwip/udp.h"
#include "lwip/tcpip.h"
#include "netif/etharp.h"
```

```
#include "lwip/dhcp.h"
#include "ethernetif.h"
#include "stm32f10x.h"
#include "arch/sys_arch.h"
#include <stdio.h>
#include "stm32_eth.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
#define MAX_DHCP_TRIES        4
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
static struct netif netif;
static uint32_t IPaddress = 0;

/* Private function prototypes -----------------------------------------------*/
static void TcpipInitDone(void *arg);
static void list_if(void);

/* Private functions ---------------------------------------------------------*/

/**
  * @brief TcpipInitDone wait for tcpip init being done
  *
  * @param arg the semaphore to be signaled
  */
static void TcpipInitDone(void *arg)
{
	sys_sem_t *sem;
	sem = arg;
	sys_sem_signal(*sem);
}

/**
  * @brief Display_IPAddress Display IP Address
  *
  */
void Display_IPAddress(void)
{
	if(IPaddress != netif.ip_addr.addr)
	{	/* IP  地址发生改变*/
		__IO uint8_t iptab[4];
		uint8_t iptxt[20];
```

```
            /* read the new IP address */
            IPaddress = netif.ip_addr.addr;

            iptab[0] = (uint8_t)(IPaddress >> 24);
            iptab[1] = (uint8_t)(IPaddress >> 16);
            iptab[2] = (uint8_t)(IPaddress >> 8);
            iptab[3] = (uint8_t)(IPaddress);

            sprintf((char*)iptxt, "     %d.%d.%d.%d       ", iptab[3], iptab[2], iptab[1],
iptab[0]);

            list_if();
      }
#if LWIP_DHCP
      else if(IPaddress == 0)
      {     // 等待 DHCP 分配 IP

            /* If no response from a DHCP server for MAX_DHCP_TRIES times */
            /* stop the dhcp client and set a static IP address */
            if(netif.dhcp->tries > MAX_DHCP_TRIES)
            {     /* 超出 DHCP 重试次数，改用静态 IP 设置 */
                  struct ip_addr ipaddr;
                  struct ip_addr netmask;
                  struct ip_addr gw;

                  dhcp_stop(&netif);

                  IP4_ADDR(&ipaddr, 10, 21, 11, 245);
                  IP4_ADDR(&netmask, 255, 255, 255, 0);
                  IP4_ADDR(&gw, 10, 21, 11, 254);

                  netif_set_addr(&netif, &ipaddr , &netmask, &gw);

                  list_if();
            }
      }
#endif
}

/**
* @brief display ip address in serial port debug windows
*/
static void list_if()
{
```

```
    printf("Default network interface: %c%c\n", netif.name[0], netif.name[1]);
    printf("ip address: %s\n", inet_ntoa(*((struct in_addr*)&(netif.ip_addr))));
    printf("gw address: %s\n", inet_ntoa(*((struct in_addr*)&(netif.gw))));
    printf("net mask   : %s\n", inet_ntoa(*((struct in_addr*)&(netif.netmask))));
}
```

在 LwIP.h 中加入：

```
#ifndef __LW_IP_H
#define __LW_IP_H

#ifdef __cplusplus
extern "C" { /* Make sure we have C-declarations in C++ programs */
#endif


/* Includes ------------------------------------------------------------------*/


/* Exported types ------------------------------------------------------------*/


/* Exported constants --------------------------------------------------------*/


/* Exported macro ------------------------------------------------------------*/


#ifndef __LW_IP_C
/* Exported variables --------------------------------------------------------*/


/* Exported functions ------------------------------------------------------- */
void Init_LwIP(void);
void Display_IPAddress(void);
void LwIP_Pkt_Handle(void);

#endif /* !defined(__LW_IP_C) */


/*
#error section
-- The standard C preprocessor directive #error should be used to notify the
programmer when #define constants or macros are not present and to indicate
that a #define value is out of range. These statements are normally found in
```

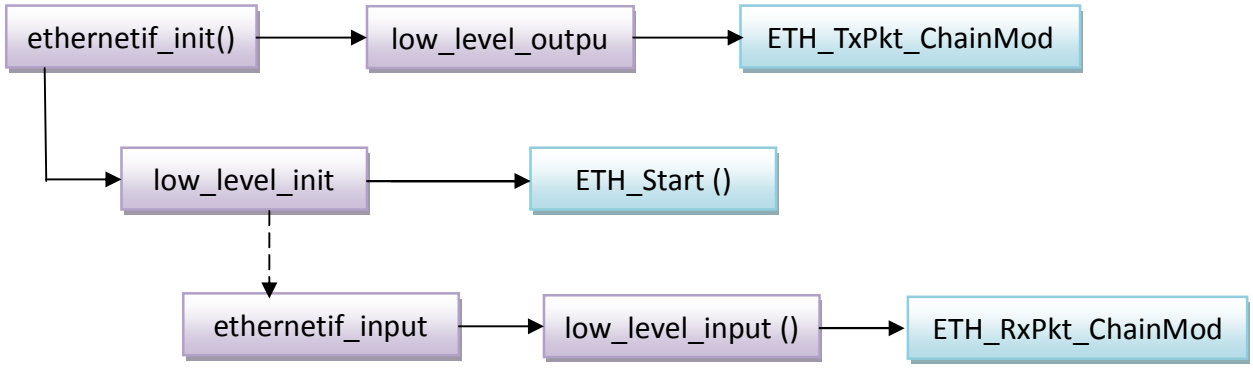
a module's .H file. The #error directive will display the message within the double quotes when the condition is not met.
*/

#ifdef __cplusplus
}
#endif

#endif /* #ifndef __LW_IP_H */
/*-- File end --*/

5.4、LwIP 硬件抽象层函数的编写

这一部分主要是完成 LwIP 硬件抽象层[HAL]ethernetif.c 的内容，它提供了硬件访问框架，真正驱动硬件的底层函数是由 stm32_eth.c 实现的。找到\LwIP\src\netif 下的 ethernetif.c 源文件，其实 LwIP 作者已经完成了大部分内容，我们只需针对我们的具体硬件做一些修改即可。
大体框架：



简单说明一下，网络接口的设置是从 ethernetif_init 开始的，low_level_init 是初始化硬件的，而真正访问硬件的是 ETH_Start[stm32_eth.c]。同样，low_level_output 和 low_level_input 分别负责发送和接受数据包，最终分别靠 ETH_TxPkt_ChainMode 和 ETH_RxPkt_ChainMode 来完成这一实质性的工作。

本着“拿来主义”的精神，我们将
\STM32F107_ETH_LwIP_V1.0.0\Utilities\lwip-1.3.1\port
下的 ethernetif.c 和 ethernet.h 拷贝到\LwIP\port 下就可以直接使用了。别忘了添加到工程目录 LwIP 的 port 组下去。

6、LwIP 配置文件 lwipopts.h

lwipopts.h 是用来配置 LwIP 的，比如说是否使用动态 ip，是否允许 TCP 服务等等，由于我们现在首要的目的是只管“马儿跑”，所以这里就简单说明了。

lwipopts.h，是在\lwip-1.3.2\src\include\lwip\opt.h 的基础上，结合 STM32F107 的一个配置文件。将\STM32F107_ETH_LwIP_V1.0.0\Utilities\lwip-1.3.1\port\lwipopts.h 拷贝到\LwIP\port 下，添加到对应的工程组下。

注意：

```
#define LWIP_DHCP                    0        //关闭动态 IP
#define SYS_LIGHTWEIGHT_PROT         0
```

7、细枝末叶

7.1、在 main.c 中调用 Init_LwIP

A、把 LwIP.h 包含到 includes.h

```
#include "LwIP.h"
```

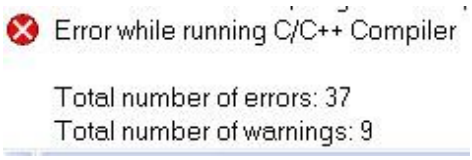
B、在 App_TaskStart()[main.c]调用 Init_LwIP ()

```
BSP_Init();
SysTick_Init();
……
Init_LwIP();
……
```

7.2、编译、链接工程

这时候可以编译一下了：

好家伙，这么多错误[具体什么错误，不一样列举分析]，



一个一个解消灭

A、

Fatal Error[Pe005]: could not open source file "arch/bpstruct.h"

```
#ifdef PACK_STRUCT_USE_INCLUDES
#  include "arch/bpstruct.h"
#endif
```

看错误提示是使用了 bpstruct.h 这个头文件，却没有找到源文件，怎么办？那新建了。下面有几个类似问题，一并解决。这儿直接拷贝了：
\STM32F107_ETH_LwIP_V1.0.0\Utilities\lwip-1.3.1\port\arch 下

拷贝到\LwIP\port\arch 在编译，OK，错误是不是少很多

Total number of errors: 25
Total number of warnings: 14

B、

接着来看，

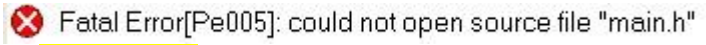
Error[Pe101]: "sys_sem_t" has already been declared in the current scope (
Error[Pe101]: "sys_mbox_t" has already been declared in the current scope

在 sys_arch.c 中的 sys_sem_t、sys_mbox_t 重复定义了，
找到 sys.h 发现，的确已经定义了，仔细的往下看，发现里面还有 sys_arch.c 的相关函数声明，与 sys_arch.h 差不多……
那怎么办？ 屏蔽掉#incude “sys.h”？为了不改变 LwIP 的源码文件，我们用一种捷径和显得更为科学的方法，事实上，LwIP 的作者也考虑到了这一问题，请看；
在 sys.h 源码开头处：
#if NO_SYS
这一条件编译，因此，我们找到 NO_SYS 的定义，在 lwipopts.h 中发现：

```
/**
 * NO_SYS==1: Provides VERY minimal functionality. Otherwise,
 * use lwIP facilities.
 */
#define NO_SYS                  1
```

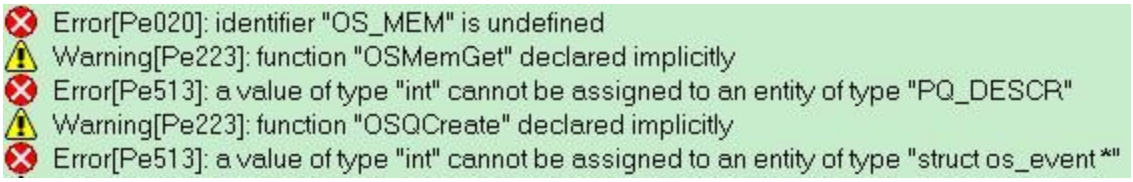
改为
#define NO_SYS　　0
编译发现：
Fatal Error[Pe035]: #error directive: "MEMP_NUM_SYS_TIMEOUT is too low to accomodate all required timeouts"
在 lwipopts.h 中将#define MEMP_NUM_SYS_TIMEOUT　　5

C、



把 ethernetif.c 中的#include "main.h"删掉，因为这里本身就没有定义这个头文件

D、



这几个问题是由于没有在 uC/OS-II 配置文件 os_cfg.h 中打开相应的开关

```
#define OS_MBOX_EN                      1
#define OS_MEM_EN                       1
#define OS_Q_EN                         1
#define OS_SEM_EN                       1
```

编译，没错误了，终于可以松口气了

到这里，是不是移植工作已经大功告成了？别急，还没完，不信？你可以试一下

E、

你还曾记得，在 LwIP.c 中有一个 **LwIP_Pkt_Handle()**的函数是用来干嘛的，看看源码就能大概明白一点点，与数据包接收有关的，只不过，在这里使用中断实现的。

但你搜索一下，发现并没有函数调用过它，好了，接下来就是干这事的：

配置 Ethernet 中断

在 BSP.c 中

```
void BSP_NVICConfiguration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    /* Set the Vector Table base location at 0x08000000 */
    NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);

    /* 2 bit for pre-emption priority, 2 bits for subpriority */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

    /* Enable the Ethernet global Interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = ETH_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
```

```
    NVIC_Init(&NVIC_InitStructure);
}
```

原型声明：

```
void BSP_NVICConfiguration(void);
```

在 BSP_Init()函数中调用它

```
……
BSP_NVICConfiguration ();
……
```

配置完中断，紧接着编写 EthernetInit 中断服务程序 ETH_IRQHandler()[stm32f10x_it.c]，在这里就调用了 **LwIP_Pkt_Handle()**函数

```
void ETH_IRQHandler(void)
{
  /* Handles all the received frames */
  while(ETH_GetRxPktSize() != 0)
  {
    LwIP_Pkt_Handle();
  }

  /* Clear the Eth DMA Rx IT pending bits */
  ETH_DMAClearITPendingBit(ETH_DMA_IT_R);
  ETH_DMAClearITPendingBit(ETH_DMA_IT_NIS);
}
```

包含 stm32_eth.h 头文件

```
#include "stm32_eth.h"
```

在 stm32f10x_it.h 中加入 ETH_IRQHandler()原型声明

```
……
void ETH_IRQHandler(void);
……
```

编译通过

8、牛刀小试

ping——有图有真相

请备好交叉线，连接 PC 和目标板

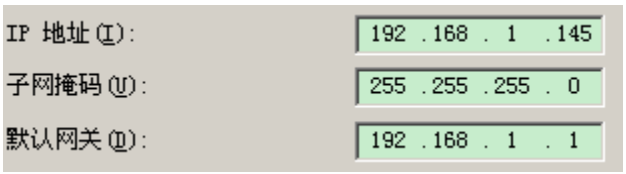
由于前面设置了

#define LWIP_DHCP　　0

因此使用的静态 IP

IP4_ADDR(&ipaddr, **192, 168, 1, 8**);

将你的 PC 设置成与之相同的网段



ping 192.168.1.8：
精彩画面呈现：

```
C:\Documents and Settings\Administrator>ping 192.168.1.8

Pinging 192.168.1.8 with 32 bytes of data:

Reply from 192.168.1.8: bytes=32 time<1ms TTL=255
Reply from 192.168.1.8: bytes=32 time<1ms TTL=255
Reply from 192.168.1.8: bytes=32 time<1ms TTL=255
Reply from 192.168.1.8: bytes=32 time<1ms TTL=255
```

谢幕

由于个人水平有限，利用业余时间编写本篇笔记，仓促之余，存在不足之处在所难免，还望指正，同时也留有不少个“为什么”，欢迎以任何方式讨论，以便继续完善。