gxTcp lib(for print) 使用指南

格芯单片机，您的网络方案专家!!!

www.tcp-mcu.com

目 录

第 1 章 gxTcp LIB(for print)使用指南

1.1 概述

随着家庭网络的兴起（即现在炒的很火的物联网），以后越来越多的设备都会纳入网络互联范围。

这就对工程师们提出新的要求，即基于 TCP/IP 协议的网络产品的开发。但是 TCP/IP 是一个庞大的协议群，协议种类非常的多，实现又非常复杂，即使是一个高级程序员要搞清楚，也非一年半载之功。再加上嵌入式领域，芯片内部的 RAM 和 ROM 等资源非常有限，需要根据各种情况对 TCP/IP 协议各项参数进行优化，就是使得网络产品的开发门槛非常之高。而且经常是即使实现了其功能，也经不起测试，一旦出问题，工程师只能对着庞大的 TCP/IP 协议代码叹气，却无从下手。

针对这种情况，格芯单片机工作室开发了 gxTcpLIB，gxTcpLIB 是一个库文件，是一个实现了 TCP/IP 协议的稳定的库文件，这样工程师不再为庞大的 TCP/IP 协议而苦恼。你可以在不需要掌握 TCP/IP 协议的情况下即可开发网络产品。

使用 gxTcpLIB 的芯片，目前已经在物联网，网络打印机，工业自动化，LED 信息显示，门禁考勤等众多领域广泛应用。

注意：gxTcpLIB 使用 ucos-ii 作为系统平台，ucos-ii 为一款开源，且免费学习的嵌入式系统，如果用于产品需要支付相关版权费用，用户如果使用 gxTcpLIB 开发产品，请自行支付 ucos-ii 相关版权费用。

1.2 gxTcpLIB 的运行平台

gxTcpLIB 需要运行在 TI 公司的 ARM CORTEX-M3 内核 MCU，我们目前主要是运行在 LM3S6911 与 LM3S9B92 两款芯片。当然，gxTcpLIB 适用于 TI 公司 ARM CORTEX-M3 内核所有集成网络功能的 MCU。

TI 公司 ARM CORTEX-M3 内核 MCU 集成网络的 MAC 和 PHY，是目前业界集成度最高的网络解决方案，其详细资源如下：

	LM3S6911	LM3S9B92
内核	32BIT ARM CORTEX-M3 架构	32BIT ARM CORTEX-M3 架构
主频	50M	80M
FLASH	256K	256K
SRAM	64K	96K
UART	3	3
ADC	无	10BIT 16 channel
CAN	无	2
Ethernet	集成 MAC and PHY	集成 MAC and PHY
是否可扩展 SDRAM	否	是
USB	无	OTG
封装	LQFP-100	LQFP-100

表 1.1 MCU 资源

由于芯片内部 MAC 和 PHY，且有合适的 FLASH 与 SRAM，所以 LM3S6911，LM3S9B92 不需要扩张任何外围即可实现网络功能，硬件原理图非常简单。

1.3 gxTcpLIB 与用户程序空间分布

如图 1.1 所示，使用 gxTcpLIB 的程序，整个 FLASH 空间被分成 3 部分，

USER CODE 为用户代码，可以用来存放用户代码和 BOOTLOAD 代码，

USER DATA 该区域可以用来保存一些用户数据，如 IP 地址等参数，

ROM LIB 起始地址是 0x3e000, 8k 空间，这部分为固化库函数，gxTcpLIB 会用到这里面的函数，如果没有这部分，gxTcpLIB 将不能正常运行。该部分被加密保护，不能被读写。

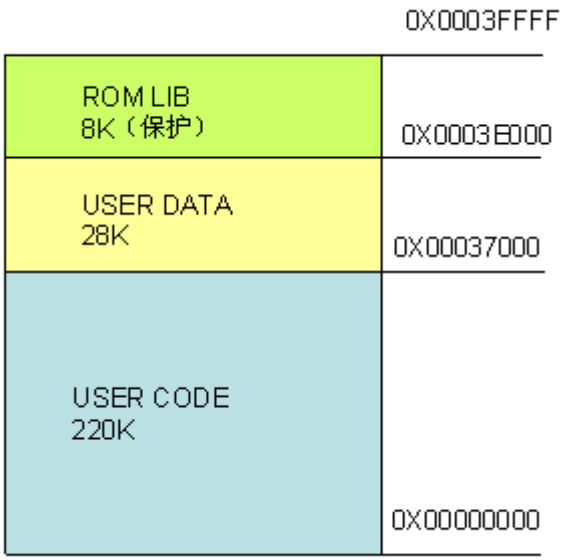

图 1.1 FLASH 空间分布

注意事项：如果是 TEMPEST 系列的芯片(如 LM3S9B92),在执行解锁操作时，会将芯片的 ROM LIB 擦除，ROM LIB 擦除后，gxTcpLIB 将不能正常运行。FURY 系列芯片（如 LM3S6911）不存在这个问题。

1.4 gxTcpLIB 工程设置

gxTcbLIB 的开发环境为 KEIL MDK。

由于 **ROM LIB** 占用高端 8K 地址空间，所以用户程序可用空间只有 248K，如图 1.2 所示，在 FLASH 空间定义里，需要将 SIZE 修改成 0x3E000

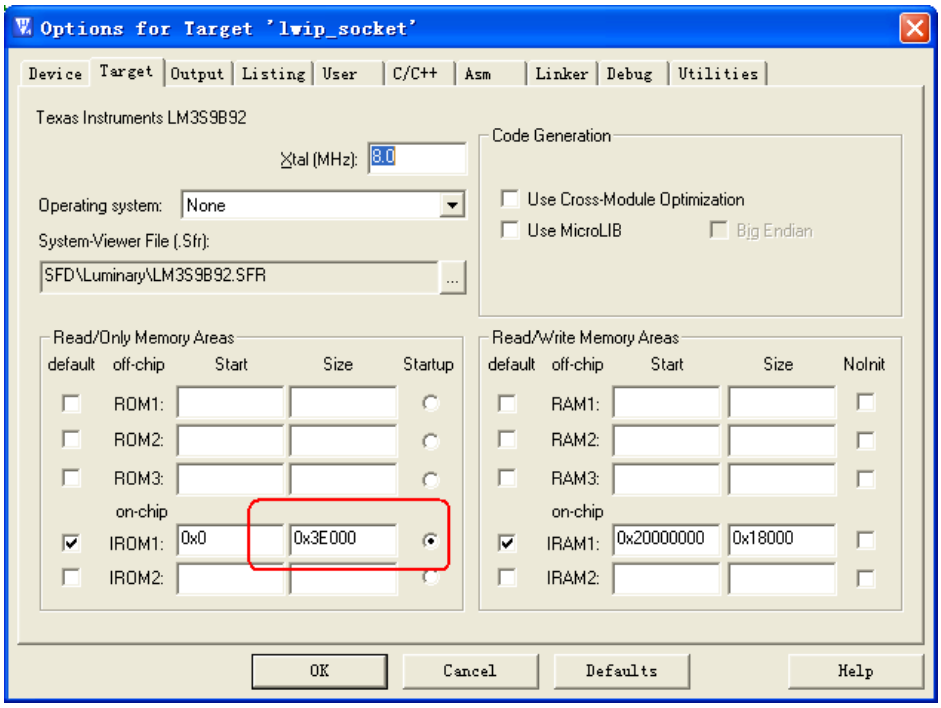

图 1.2 设置 FLASH 空间

如图 1.3 所示，使用默认的分散加载设置

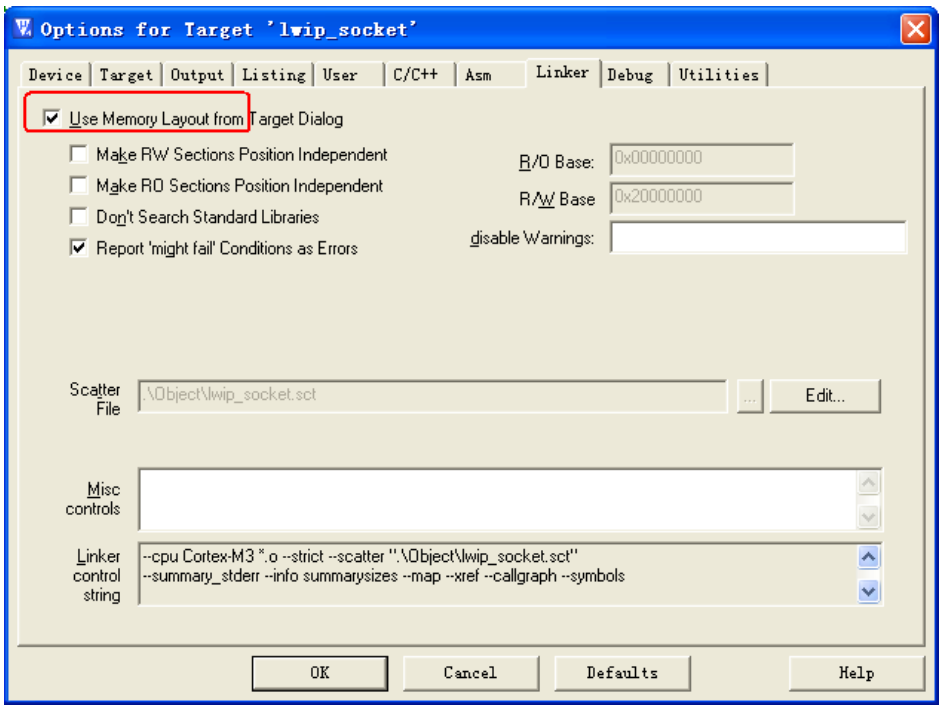

图 1.3 使用默认分散加载

1.5 工程目录结构

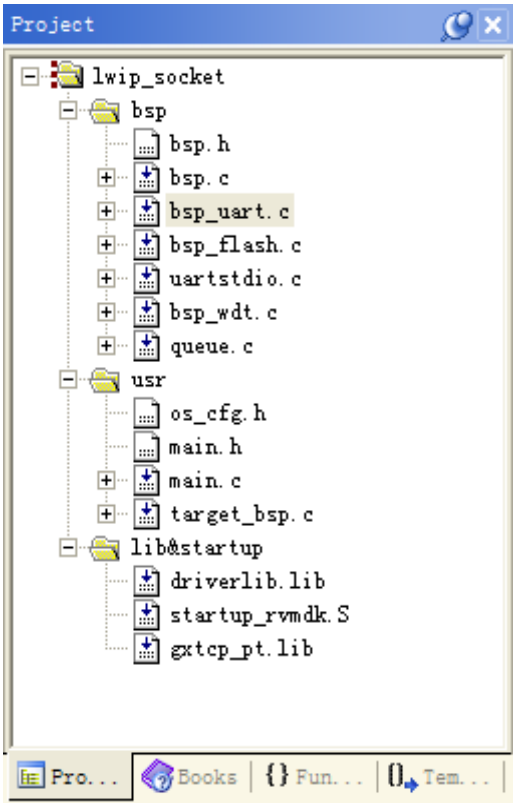

图 1.4 工程目录结构

如图 1.4 所示，gxTcp LIB 应用工程主要由 bsp, usr, lib&startup 3 个文件夹组成。

bsp (board software pack)文件夹，顾名思义，即目标板软件开发支持包，主要包括目标板硬件驱动功能组件和一些通用功能组件。

usr 文件夹，主要放置 main.c 和用户应用程序。

lib&startup 文件夹，包括了 TI M3 外设驱动库——driverlib.lib；TI M3 启动配置文件——startup_rvmdk.s 和我们的 TCP/IP 应用库——gxtcp_pt.lib

1.6 工程文件介绍

接下来我们开始对 gxTcp LIB 应用工程的文件进行详细介绍。

1.6.1 UART 驱动功能组件 bsp_uart.c

bsp_uart.c 为 UART 驱动功能组件，且这是一个多任务下的 UART 驱动功能组件。

UART 驱动功能组件有一个自己独立的接受缓冲池。当 UART 发送接受中断时，UART 中断服务程序将数据从接受 FIFO 移入申请接受缓冲块；当调用 Uart_Rcv（）函数时，即可取出内存块中的数据。UART 接受缓冲池内存块的数目和内存块的大小是可以修改，如程序清单 1.1， UART_RCV_BLOCK_SIZE 定义了内存块的大小，UART_MSG_NUM 定义了内存块的数目。

```
#define     UART_RCV_BLOCK_SIZE     200     // UART 接受缓冲块大小
#define     UART_MSG_NUM            12      // UART 接收发送消息,最大条数
```

程序清单 1.1 定义 UART 接受缓冲

函数名称：int32 Uart_Open(STRUCT_UART *uart)
函数功能：对相应 UART 进行初始化，
输入参数：uart ---- UART 配置结构体指针。
返回值： 函数执行状态，0 --- 成功初始化

函数名称：int32 Uart_Rcv(STRUCT_UART *uart , uint16 headtime , void *rcvdata)
函数功能：从 UART 缓冲池中取数据。如果 UART 缓冲池中有数据，该函数立马返回数据 UART 接受的数据长度；如果 UART 缓存池中没有数据，该函数会阻塞等待数据，当等待时间超过设定超时时间时，该函数会返回 0。
输入参数：uart ---- UART 配置结构体指针。
headtime ---- 接受超时时间。
Rcvdata ---- 接收数据目的地址。
返回值：数据长度。

函数名称：uint8 Uart_Send(STRUCT_UART *uart , uint8 *pucBuffer, uint32 ulCount)
函数功能：将数据发送到 UART
输入参数：uart ---- UART 配置结构体指针。
pucBuffer ---- 需要发送的数据源地址
ulCount ---- 发送数据长度

函数名称：void UART0IntHandler(void)
函数功能：UART0 中断服务程序。该函数需要添加到 startup_rvmdk.s 文件的中断向量表中。UART1IntHandler， UART2IntHandler 分别为 UART1,UART2 的中断服务程序。

1.6.2 看门狗功能组件 bsp_wdt.c

bsp_wdt.c 为 WTD 驱动功能组件，且这是一个多任务下的看门狗功能组件。

如程序清单 1.2 所示，通过修改 MAX_DOG_NUM 的数值，即可定义相应数目的看门狗。如程序清单 1.2 所示，定义了 4 条看门狗，每个独立的看门狗都可以单独的使能，禁能，可以在 4 个不同的任务中起到防止程序进入死胡同的功能。

注意：看门狗 0，看门狗 1， 保留给 gxTcp LIB 使用，用户应用程序只能用编号为 2 以上的看门狗。

```
#define MAX_DOG_NUM         4
```

程序清单 1.2 定义看门狗数目

函数名称：void WD_HW_Init (uint32 cycle)

函数功能：初始化看门狗设置，并设置看门狗复位时间，如果超过设定时间，应用程序没有喂狗，看门狗将复位 MCU。

输入参数：cycle ----门狗复位时间

函数名称：void WD_Enable(int no)

函数功能：因为每条看门狗，都可以单独使能或禁能，该函数使能看门狗复位功能，看门狗只有在使能后才能发挥看家本领。

输入参数：no ---- 看门狗编号

函数名称：void WD_Disable(int no)

函数功能：因为每条看门狗，都可以单独使能或禁能，该函数禁能看门狗复位功能，禁能后，看门狗将不能起到看家的作用，注意看门狗功能模块在初始化的时候，默认把所有看门狗禁能。

输入参数：no ---- 看门狗编号

1.6.3 串口字符串打印功能组件 uartstdio.c

TI 有提供一个类似标准 C 中 prinf()函数的功能组件，该组件提供了函数 UARTprintf（），实现 print()函数一样的功能，可以完成数据的十进制字符串，十六进制字符串转换。

UARTprintf（）函数数据默认打印窗口为 UART0，在调用调用初始化函数 UARTStdioInit（）时设置。

函数名称：void UARTStdioInit(unsigned long ulPortNum)

函数功能：UARTprintf（）函数在使用前，必须调用本函数进行初始化。
调用本函数后，对应 UART 的波特率设置为默认的 115200

输入参数：ulPortNum --- UART 端口序列号， 0 代表使用 UART0 作为数据打印窗口。

函数名称：void UARTprintf(const char *pcString, ...)

函数功能：字符串打印，与标准 C 中 printf()函数功能完全一样，具体用法请参考 printf()函数。

1.6.4 内部 FLASH 读写功能组件 bsp_flash.c

bsp_flash.c 为内部 FLASH 读写功能组件。

TI-M3 TEMPEST 系列 MCU 的内部 FLASH 在一些极端情况下，擦写次数只有几百次，为了解决这个问题，我们加入了一个写平均算法。简单地说就是将一份数据申请 10 份的空间，在擦写 FLASH 时，平均地使用这 10 份空间，这样相当于将 FLASH 的使用次数提高了 10 倍。当然如果 FLASH 的寿命只有几百次，提高 10 倍也才几千次，所以如果用户需要要使用 TEMPEST 系列 MCU 芯片内部 FLASH 存储数据存储器时，必须确保两点：

1. 数据的擦写次数不是很频繁，总次数在 1 万次以下
2. 数据量不是很大，因为一个数据，我们是用 10 倍的空间来做算法的

如程序清单 1.3 所示，在 bsp_flash.h 中定义了数据存储区的起始地址。

程序清单 1.3 FLASH 数据区起始地址

```
#define     INITAL_FLG_ADDR                 (210*1024)
```

函数名称：void InitalFlash(void)
函数功能：初始化 FLASH 设置，如果是第一次上电，擦除所有数据区域。

函数名称：void SaveFlashData(unsigned long *pulData, unsigned long ulAddress, unsigned long ulCount)
函数功能：将数据保存进 FLASH，带写平均算法。
输入参数：pulData --- 操作的数据缓冲区
ulAddress --- 写 FLASH 起始地址
ulCount --- 写 FLASH 长度，单位 BYTE

函数名称：unsigned long GetSaveData(unsigned long *pulData, unsigned long ulAddress, unsigned long ulCount)
函数功能：从 FLASH 中获得保存的数据
输入参数：pulData --- 操作的数据缓冲区
ulAddress --- 写 FLASH 起始地址
ulCount --- 写 FLASH 长度，单位 BYTE

1.6.5 数据队列 queue.c

格芯单片机有提供给用户一个灵活、方便的数据队列功能模块。数据队列初始化后，即可方便的进行入队，出队操作。

数据队列在使用时需要申请一块内存空间，如程序清单 1.4 所示，一般可以定义一个全局数组。

程序清单 1.4 数据队列空间

```
uint8 queue_buf[MAX_RCV_NUM];
```

如程序清单 1.5 所示，我们可以设置数据队列空间的大小，打印接受需要开辟一个大的缓冲空间，建议一般在 20K 以上。

程序清单 1.5 数据队列空间大小

```
#define MAX_RCV_NUM                24000                    // 定义缓冲大小
```

当然该数据队列可以使用 SDRAM 中的内存空间,如程序清单 1.6 所示，只需在初始化队列时，将使用一个指向 SDRAM 空间的指针即可。

程序清单 1.6 数据队列使用 SDRAM 空间

```
uint8 *queue_buf;
queue_buf=0x60000000;
queue_inital(queue_buf, 1000000);
```

函数名称： void queue_inital(void *buf, unsigned long buf_size)
函数功能： 队列初始化函数，队列使用之前必须初始化。
输入参数： buf --- 队列缓冲地址
size --- 为初始化时，传递给队列的缓冲区的最大长度

函数名称：long queue_in(void *buf, unsigned char *str, unsigned long len)
函数功能： 数据入队。
输入参数： buf --- 队列缓冲地址
str --- 入队数据指针
len --- 入队数据长度

函数名称： long queue_out(void *buf, unsigned char *pbuf,
unsigned long rcvlen, unsigned long timeover)
函数功能： 数据出队。
输入参数： buf --- 队列缓冲地址
pbuf --- 数据出队目的地址
rcvlen ---- 出对长度
timeover --- 超时时间，如果在指定时间队列中数据不够指定的出队长度
rcvlen，函数 queue_out 会超时退出，返回值为超时时出队数据长度。
当 timeover 为 0 时，代表函数 queue_out 永不超时退出。

1.6.6 其他组件

bsp.c 为一些系统常用函数。

target_bsp.c 为与目标板硬件相关的配置。

main.c 里放置了主程序入口，与启动任务，另外，可以在这里添加用户应用 。

driverlib.lib 为 TI 外设驱动库。

gxTcp_pt.c 为 TCP/IP 库。

startup_rvmdk.S 为 MCU 启动汇编程序与中断向量表。

1.7 gxTcpLIB 运行

正确连接好硬件与仿真工具，打开 gxTcp 工程，点击 DEBUG 按钮  ，即可开始调试仿真你的程序。

如果程序你正常运行到 target_bsp.c 文件中 InitBSP（）函数中，UARTprintf("APP Start!\r\n")语句，打印 APP Start，说明网络硬件初始化成功，可以开始你的应用程序。

```
229     //=====================================
230     // 初始化TCP/IP协议
231     //=====================================
232     i = InitNic();
233     if (i)
234     {
235         UARTprintf("ERROR: %d\r\n", i);
236         while(1);
237     }
238     UARTprintf("APP Start!\r\n");
239
```

图 1.5 网络硬件初始化程序

1.8 gxTcpLIB 中断优先级分配

TI-M3 硬件支持 8 级中断嵌套，函数 IntPrioritySet(unsigned long ulInterrupt, unsigned char ucPriority)可以设置中断优先级，ulInterrupt 为中断号， ucPriority 为优先级。

注意：ucPriority 高 3 位有效，所以需要将优先级左移 5 位。且数值越小，代表优先级越高。

如程序清单所示，默认所有中断优先级为 1，或大于 1，优先级 0 保留给要求快速响应的特殊应用，优先级 0 保留给要求快速响应的特殊应用。

程序清单 1.7 系统中断优先级分配

```
IntPrioritySet(FAULT_PENDSV, (1<<5));          // 系统软中断
IntPrioritySet(FAULT_SYSTICK, (1<<5));         // 系统定时器中断
IntPrioritySet(INT_WATCHDOG, (1<<5));          // 看门狗中断
IntPrioritySet(INT_UART0, (1<<5));             // UART0 中断
IntPrioritySet(INT_ETH, (1<<5));               // Ethernet 中断
```

1.9 gxTcpLIB 数据结构

gxTcpLIB 的初始化参数，如 IP 地址，网关，通讯端口等，保存在结构体 ModuleCfg 中，其定义如程序清单 1.8 所示。

IP, MASK, GATEWAY 等常用参数我就不再一一介绍，下面我重点介绍以下几个参数：

over_time 大于 0，代表 TCP 链接超时时间，如果 TCP 链接超过设定时间没有接收到网络数据包，gxTcpLIB TCP 应用进程会主动断开 TCP 链接；

over_time 等于 0，代表 TCP 应用进程永远不会主动断开 TCP 链接。

en_wdt 等于 1，代表 gxTcpLIB TCP 应用进程开启看门狗复位功能；

en_wdt 等于 0，代表 gxTcpLIB TCP 应用进程没有开启看门狗复位功能。

gxTcpLIB 开启了任务堆栈统计功能，当 en_pt_stk 等于 1，系统统计任务会在串口调试台打印堆栈使用情况。en_pt_stk 一般默认为 0。当我们需要观看堆栈使用情况是，在 TCP 的 4000 端口，输入 es 指令即可设置 en_pt_stk 等于 1。

在调试程序时为了方便，建议关闭看门狗与超时主动断开 TCP 链接功能；在产品出厂时为了提供系统的可靠性，建议开启看门狗和超时主动断开 TCP 链接功能。

程序清单 1.8 模块参数结构体

```
STRUCT_IP_CFG     ModuleCfg;                          // 模块参数结构体
typedef __packed struct
{
    unsigned char     Pwd[8];                 // 密码
    unsigned long     DevID;                  // 模块出厂序列号
    unsigned char     Mac[8];                 // MAC 号， 最后两位无效
    STRUCT_IP_CLASS IPs;                      // IP 地址结构
    //=========================================================================
    //=========================================================================
    unsigned long     port;                   // 主通讯端口
    unsigned long     over_time;              // 超时时间，如果超时，模块主动断开 TCP 链接
    unsigned char     en_wdt;                 // 使能看门狗
    unsigned char     en_pt_stk;              // 使能打印堆栈信息
    unsigned char     recv[2];                // 为字节对齐，保留 3 字节

} STRUCT_IP_CFG   ;
```

InitNic() 函数调用结构体 ModuleCfg 中的参数，初始化网络。

ModuleCfg 参数默认保存在 FLASH 中，在上电后，调用函数 GetModuleConfig()获得。

1.10 TCP 数据的接受流程

网络初始化完成后，gxTcpLIB 会创建一个 TCP APP Task 的任务，该任务从指定端口接受数据,例如打印机的数据接端口为 9100。

TCP 数据接受流程如图 1.6 所示。

1. 准备工作，初始化系统与网络，初始化队列，队列的初始化对象为指针 queue_buf，queue_buf 可以是一个数组，也可以是一个指向某一内存块的指针。

2. 网络中断服务程序接受物理层过来的数据。

3. TCP APP Task 的任务接受 TCP 链接的数据，并调用函数 queue_in()数据入队，队列操作对象为 queue_buf 指针。

4. User Task 任务调用出队函数 queue_out（），获得网络接受的数据。

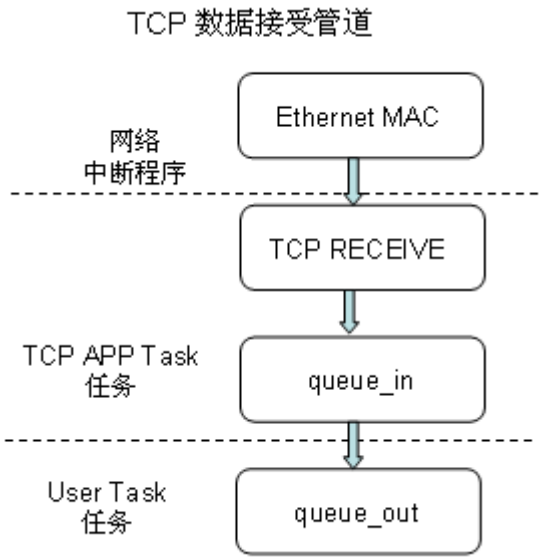

图 1.6 TCP 数据接受流程

1.11 任务与任务优先级

uC/OS-II 是一个多任务系统，也是一个抢占型系统，通过有先级仲裁，系统总是运行最高就绪态的最高优先级任务。

所以高优先级的任务在处理完相应事件后，必须调用系统的延时函数，释放 MCU 的使用权，否则低优先级的任务将得不到运行。

uC/OS-II 系统数字越小，优先级越高，0 为最高优先级任务。

gxTcpLIB 基于多任务的操作系统 ucos-ii, 如表 1.2 所示，gxTcpLIB 一共有 9 个任务。

表 1.2 系统任务列表

任务	优先级	描述
uC/OS-II Stat	62	系统统计任务，统计堆栈使用情况等，如果 en_pt_stk==1，将在 UART 打印堆栈使用情况。
uC/OS-II Idle	63	系统空闲任务，也是最低优先级任务，当系统所有任务都处于等待状态时，系统将在空闲任务里调用休眠指令，以降低功耗。
tcpip_thread	2	TCP/IP 协议处理任务,在函数 InitNic() 中创建。
TCP_PACK Task	5	TCP/IP 协议处理任务,在函数 InitNic() 中创建。
Start Task	8	启动任务，在系统启动时初始化系统，创建其他任务，创建完其他任务后一直挂起。
TCP APP Task	9	TCP&UDP 应用任务，创建 TCP 链接，接受 9100 端口 TCP 数据，然后入队。
Query status	10	TCP&UDP 应用任务，用于查询打印机状态，设置一些系统参数，通讯端口为 4000。
UDP FIND Task	15	TCP&UDP 应用任务，UDP 链接，用于广播查询 IP 地址。
Send to Print	20	用户任务，从队列里取数据，并将出对数据发送给打印机或 UART。用户可以修改此任务，用于应用程序处理。

如果用户应用程序需要增加任务，或进行任务优先级调整，请遵守以下规则：

1. 优先级 63,62 为 ucos 系统任务，不能动。
2. 优先级 10 以下任务保留给 TCP 协议处理与启动任务。
3. 优先级 10--19 保留给 TCP 与 UDP 应用程序处理。
4. 用户任务优先级可以设置在 20--60 之间。

TCP/IP 协议处理任务与 TCP&UDP 应用任务都是基于信号量阻塞机制，当网络上没有数据包时，这些任务都处于挂起状态，不占用 CPU 的使用权。

1.12 任务堆栈与堆栈测试

每个任务都需要一个堆栈，堆栈空间分配是个很头痛的问题，堆栈开大一点吧，RAM 的资源非常有限和宝贵，怕浪费；堆栈开小一点吧，就怕堆栈溢出，导致系统崩溃。所以格芯单片机在创建任务时，先是给堆栈开一个较大空间，然后做堆栈测试，根据测试的堆栈使用情况再调整堆栈大小。

堆栈测试的一般原则是，先把所有的功能都运行一次，然后进行海量数据测试与长时间测试，这时得到的堆栈使用数据再上浮 30%，为最理想的堆栈大小。

gxTcpLIB 默认开启了堆栈测试功能，并且通过函数 StatInfoPrint()，将堆栈使用情况打印到串口。当然虽然默认开启了堆栈测试功能，但是不会往串口输出，如果需要输出，需要在 TCP 链接 4000 端口，连接并发送“es”字符串，使能堆栈测试输出。堆栈测试输出使能后，就可以通过串口数据看到堆栈的使用情况了。

格芯单片机提供的 gxTcpLIB 是做过堆栈测试的，测试数据如下：

```
tcpip_thread    PRIO: 2     STK used: 106   STK free: 150
TCP_PACK Task   PRIO: 5     STK used: 134   STK free: 122
Start Task      PRIO: 8     STK used: 88    STK free: 40
TCP APP Task    PRIO: 9     STK used: 360   STK free: 60
Query status    PRIO: 10    STK used: 114   STK free: 50
UDP FIND Task   PRIO: 15    STK used: 104   STK free: 60
Send to Print   PRIO: 20    STK used: 60    STK free: 136
uC/OS-II Stat   PRIO: 62    STK used: 59    STK free: 21
uC/OS-II Idle   PRIO: 63    STK used: 19    STK free: 31
```

1.13 系统状态查询与参数设置指令

我们前面有提到创建一个任务 Query status，该任务用于查询打印机状态，设置一些系统参数，TCP 通讯端口为 4000。

表 1.3 系统参数设置命令

指令	二进制码	功能描叙
0x1b v	0x1b 0x76	打印机状态查询指令。如果 Query status 在创建任务时，有设置打印机状态查询回调函数，则执行回调函数，并返回打印机状态；如果 Query status 在创建任务时，没有设置打印机状态查询回调函数，则返回状态信息无效。
es	0x65 0x73	enable print stk information 的缩写，使能堆栈信息输出，执行此命令后，设置 en_pt_stk==1，UART 窗口会有堆栈使用信息输出。
ds	0x64 0x73	disable print stk information 的缩写，禁能打印堆栈信息，执行此命令后，设置 en_pt_stk==0。
ed	0x65 0x64	enable watch dog 的缩写，使能 TCP APP Task 任务的看门狗，执行此命令后，设置 en_wdt==1。
dd	0x64 0x64	disable watch dog 的缩写，禁能 TCP APP Task 任务的看门狗，执行此命令后，设置 en_wdt==0。
st=ot	0x73 0x74	set over time 的缩写，设置 TCP APP Task 任务创建主通讯端口链接超时时间，设置 over_time = ot，单位为 100ms。如果设置命令为“st=1000”,即设置超时时间为 100 秒。如果主通讯端口在建立链接后，超过超时时间没有收到 TCP 数据，TCP APP Task 任务会主动关闭链接；如果 over_time==0， TCP APP Task 任务不会主动关闭链接。