分类号：
密　级：

Y968700

学校代码：10055
学　　号：030263

南開大學
Nankai University

论文题目：TCP/IP协议在嵌入式系统上的实现

硕士学位论文
MASTER'S DISSERTATION

培养院系：信息技术科学学院
一级学科：信息与通信工程
二级学科：通信与信息系统
论文作者：康小军
指导教师：张立民　副教授

九公九能
日新月異

南开大学研究生院

2006年5月

摘 要

嵌入式操作系统μC/OS-II以其精简的内核和高实时性以及可靠性、裁剪性在工业控制领域内得到了广泛应用。μC/OS-II操作系统装配于增强型51微处理器上整合TCP/IP网络协议栈后成为了一个完整的开发平台，另外，基于此平台的开发也将使软件缺陷大幅度减少，从而可以开发更为稳定的应用软件。为此首先需要使硬件系统工作正常，微控器可以控制网卡芯片的数据收发，然后把μC/OS-II这个微内核的嵌入式操作系统移植到该硬件平台上，本论文重点是分析研究该系统原理，移植细节，以及如何在这个嵌入式系统上做开发应用工作。在μC/OS-II操作系统下编ARP、IP、TCP、UDP等协议。

本论文实现的是网络中Server端的功能。在协议栈中共建立了三个任务：网卡接收任务，实现查询网卡，判断是否有数据到达；TCP-Server任务，与客户端建立TCP连接；UDP-Server任务，与客户端建立UDP连接。三个任务中，TCP-Server任务的优先级最高，三者通过μC/OS-II中提供的信号量功能实现任务之间的切换。上述主要功能和算法均由软件编程实现小型TCP/IP协议栈。硬件系统的容量是完全可以满足的，而且μC/OS-II的通用性非常好，所以可方便地进行算法升级，同时也便于移植，硬件系统所提供的ROM和RAM空间小型对于小型协议栈是足够的。

关键词：嵌入式μC/OS-II操作系统；TCP/IP协议；信号量；驱动程序

Abstract

Embeded operating system μ C/OS-II has been applied extensively in the field of industrial contral because of its high real-time quality and dependability simplified kernel. It is easy to understand and promot. In this atical, writer probes into the TCP/IP protocol and μ C/OS-II operation system, and compile TCP/IP protocol stack with μ C/OS-II OS.The platform equipped with μ C/OS-II and TCP/IP stack becomes a developing platform.Programmers can use APIs and services of μ C/OS-II. They can also use network functionalities for their application. By this way, it will decrease the occurrence of bugs in applications.

In this system' s hardware, the main CPU is 51 micro-controller; ethernet card is RTL8019AS FROM TEALTEK SEMI-CONDUCTOR CO.LTD. In the solution, the primary task is to make hardware system work smoothly, and micro-controller can handle data transfer in ethernet card. Then, transfer embedded OS into the micro-controller. Compile ARP, IP, UDP and TCP protocol by μ C/OS-II OS.

This solution makes the function of Server end in network into reality. There are three tasks in the protocol stacks: ethernet card reception, which can inquire whether the data arrived in destination or not; TCP-Server, which is set up TCP connection with client end; UDP-Server, which is set up UDP connection with client end. The task of TCP-Server' s priority is the highest among these three tasks. The tasks can be switched via semaphore provided by μ C/OS-II. Above mentioned ones is the normal process in the solution. For specific process, memory allocated parameter and data buffer should be defined per specific application. Above mention function and algorithm are completed with software compilation. It is easy to update algorithm and convenient to transfer because no change in hardware. This project shows the ROM and RAM space of hardware is enough for mini protocol stack.

Key word: μ C/OS-II embedded operation system; TCP/IP protocol; semaphore; driver application

南开大学学位论文版权使用授权书

本人完全了解南开大学关于收集、保存、使用学位论文的规定，同意如下各项内容：按照学校要求提交学位论文的印刷本和电子版本；学校有权保存学位论文的印刷本和电子版，并采用影印、缩印、扫描、数字化或其它手段保存论文；学校有权提供目录检索以及提供本学位论文全文或者部分的阅览服务；学校有权按有关规定向国家有关部门或者机构送交论文的复印件和电子版；在不以赢利为目的的前提下，学校可以适当复制论文的部分或全部内容用于学术活动。

学位论文作者签名：康小军

2006 年 5 月 28 日

经指导教师同意，本学位论文属于 保密，在 年解密后适用本授权书。

指导教师签名：	张论文	学位论文作者签名：	康小军
解 密 时 间：	年 月 日		

各密级的最长保密年限及书写格式规定如下：

内部 5 年（最长 5 年，可少于 5 年）
秘密★10 年（最长 10 年，可少于 10 年）
机密★20 年（最长 20 年，可少于 20 年）

南开大学学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：康小军

2006 年 5 月 27 日

第一章 绪论

第一节 课题提出的背景和意义

由于 Internet 在世界范围内的大量普及应用，因此Internet 所使用的TCP/IP协议也就成为使用最广泛的协议。同时它也在智能小区，远程控制等多方面得到广泛的应用。各种各样的网络设备的出现为人们的生活提供了很多的方便。而这些网络设备的核心就是嵌入式 TCP/IP 协议栈，它为设备的上网提供了强有力的支持。目前能够实现嵌入式 TCP/IP 协议栈的操作系统有很多，比如 Vxworks, Linux, Psos, μC/OS 等。这些操作系统各有各自的优缺点，单纯从经济上考虑 Linux 和μC/OS 是最有竞争力的，因为它们是免费的。尤其是μC/OS，它的内核精简，而且源码完全公开，这也是选用它的一个原因。

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，已经有了很多的应用，是一个经实践证明稳定、可靠、好用的内核。μC/OS-II 操作系统与其它常见的嵌入式系统相比，如 WinCE,VxWorks,pSos 等，除μC/OS-II 自身的特点外，还因为对系统硬件资源需求非常低，可以在许多低端处理器甚至是 8 位单片机系统上运行，从而有效降低采集节点成本。这些特点是许多常见商用操作系统都不具备的。而且μC/OS-II 开放源代码，开发授权费用低廉，这些优势也使其更适合嵌入式工控设备开发。

51 系列微处理器基于简化的嵌入式控制系统结构，被广泛应用于从国防，自动控制到 PC 机上的键盘上的各种应用系统上。这些增强型芯片给 51 系列单片机加入了更高的性能和外部功能，像 I^2C 总线接口，模拟量到数字量的转换，看门狗，PWM 输出等。不少芯片的上作频率达到 40MHz，工作电压下降到 1.5V，由于这些功能都是基于同一个内核，使得 51 系列单片机很适合作为工业控制产品的基本构架。51 系列单片机能够运行各种程序，本文的研究意义在于向 51 硬件系统上植入了操作系统和协议栈后，其后续开发就将在操作系统上进行，开发人员只需调用操作系统提供的 API 以及各种服务，而无需自己去实现各种底

层功能，进而可以减少引入软件缺陷的机会。另外，所开发的应用可以通过对协议栈 API 的调用获得完整的网络功能，使其成为一个标准的成本低廉的网络终端。

第二节 本论文的主要研究工作

本论文将沿着将 TCP/IP 协议移植到嵌入式系统这条主线，对涉及的相关软硬件以及移植的相关技术进行深入研究，主要的研究内容是:

- TCP/IP 协议的分析;
- μC/OS-Ⅱ嵌入式操作系统的分析和移植，并且成功实现其在 51 微控器上的移植、运行;
- 实现 51 微控器控制 RTL8019 网络芯片编写小型 TCP/IP 协议栈;
- 使用μC/OS-Ⅱ操作系统在目标板上实现 TCP/IP 协议栈,完成网络中 Server 端的功能，实现 TCP，UDP 的连接。

上述工作完成了从基础知识准备，到对移植的相关对象(包括硬件及软件)的了解，到对代码的阅读及改写，到调试和测试驱动的编写等全过程。从更宽的角度来看，是软件向某个特定硬件系统的移植全过程，而不仅是将目标局限在 51 硬件系统上，还展示在各种移植过程中的共性。论文于 2005 年开始着手进行，到目前为止，已经基本完成预定目标，实现了既定的功能。

第二章 TCP/IP 协议与局域网协议

TCP/IP（Transmission Control Protocol/Internet Protocol 的简写，中文译名为传输控制协议/互联网络协议）协议是 Internet 最基本的协议，简单地说，就是由底层的 IP 协议和 TCP 协议组成的。参考开放系统互连（OSI）模型，TCP/IP 通常采用一种简化的四层模型，分别为：应用层、传输层、网络层、链路层。[1]

第一节 TCP/IP 协议体系结构

图 2.1 为 TCP/IP 与 OSI 这两种体系结构的对比。TCP/IP 一开始就考虑到多种异构网的互连问题，并将网际协议 IP 作为 TCP/IP 的重要组成部分。IP 协议在网络层中，处理分组在网络中的活动，例如分组的选路等。在 TCP/IP 协议族中，网络层协议包括 IP 协议（网际协议），ICMP 协议（Internet 互联网控制报文协议），以及 IGMP 协议（Internet 组管理协议）。[2]

<table>
<tr><th>OSI参考模型
层描述</th><th>OSI
层号</th><th>TCP/IP层
描述</th></tr>
<tr><td>应用层</td><td>7</td><td rowspan="3">进程/应用层</td></tr>
<tr><td>表示层</td><td>6</td></tr>
<tr><td>会话层</td><td>5</td></tr>
<tr><td>传输层</td><td>4</td><td>主机到主机层</td></tr>
<tr><td>网络层</td><td>3</td><td>网际层</td></tr>
<tr><td>数据链路层</td><td>2</td><td rowspan="2">网络访问层</td></tr>
<tr><td>物理层</td><td>1</td></tr>
</table>

图 2.1 TCP/IP 与 OSI 两种体系结构对比

TCP/IP 一开始就对面向连接和无连接的服务并重。这由在运输层中的 TCP（传输控制协议）和 UDP（用户数据报协议）。两个协议来保证。TCP 为两台主机提供高可靠性的数据通信（面向连接的服务）。它所做的工作包括把应用程序交给它的数据分成合适的小块交给下面的网络层，确认接收到的分组，设置发送最后确认分组的超时时钟等。由于运输层提供了高可靠性的端到端的通信，

因此应用层可以忽略所有这些细节。而另一方面，UDP 则为应用层提供一种非常简单的服务（无连接的服务）。几乎各种不同的 TCP/IP 实现都会提供下面这些通用的应用程序：Telnet 远程登录；FTP 文件传输协议；SMTP 简单邮件传送协议；SNMP 简单网络管理协议。

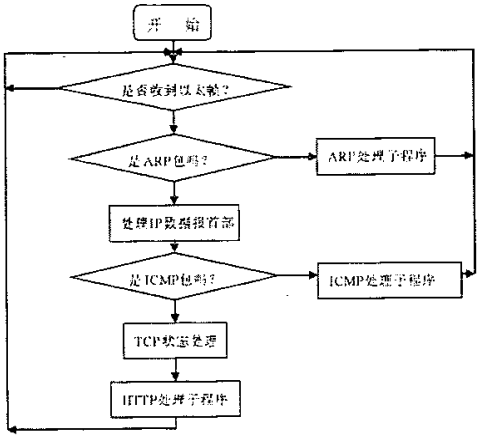

图 2.2 系统 TCP/IP 协议部分程序流程图

假设在一个局域网（LAN）如以太网中有两台主机，二者都运行 FTP 协议，图 2.2 列出了该过程所涉及到的所有协议。

图 2.2 这里，列举了一个 FTP 客户程序和另一个 FTP 服务器程序。大多数的网络应用程序都被设计成客户—服务器模式。服务器为客户提供某种服务，在本例中就是访问服务器所在主机上的文件。在远程登录应用程序 Telnet 中，为客户提供的服务是登录到服务器主机上。在同一层上，双方都有对应的一个或多个协议进行通信。例如，某个协议允许 TCP 层进行通信，而另一个协议则允许两个 IP 层进行通信。

第二节　局域网体系结构

2.2.1　IEEE 802 参考模型[3]

局域网的体系结构与广域网的有相当大的区别。由于局域网只有一个计算机通信网，而且局域网不存在路由选择问题，因此它不需要网络层，而只有最低的两个层次。然而局域网的种类繁多，其媒体接入控制的方法也各不相同，远远不像广域网那样简单。为了使局域网中的数据链路层不至过于复杂，就应

当使局域网的数据链路层划分为两个子层，即：媒体接入控制或媒体访问控制 MAC(Medium Access Control)子层和逻辑链路控制 LLC(Logical Link Control)子层，而网络的服务访问点 SAP 则在 LLC 层与高层的交界面上。

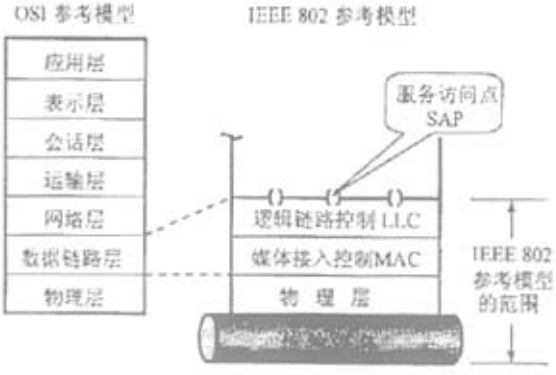

图 2.3　局域网的 802 参考模型与 OSI 对比[4]

图 2.3 这其中物理层的主要功能是：信号的编码与译码；为进行同步用的前同步码的产生与去除；比特的传输与接收。与接入各种传输媒体有关的问题都放在 MAC 子层。MAC 子层还负责在物理层的基础上进行无差错的通信。更具体些讲，MAC 子层的主要功能是：将上层交下来的数据封装成帧进行发送(接收时进行相反的过程)；实现和维护 MAC 协议；比特差错监测；寻址。

2.2.2　逻辑链路控制 LLC 子层[5]

一个主机中可能有多个进程在运行，它们可能同时与其他的一些进程(在一个主机或多个主机中)进行通信。因此，在一个主机的 LLC 子层上面应设有多个服务访问点，以便向多个进程提供服务。可见在网络中的进程通信时，需要有以下两种地址：

(1)MAC 地址，即主机在网络中的站地址或物理地址，这由 MAC 帧负责传送；

(2)SAP 地址，即进程在某一个主机中的地址，也就是 LLC 子层上面的服务访问点 SAP，这由 LLC 帧负责传送。

这就是说，网络中的寻址要分两步走。第一步是用 MAC 帧的地址信息找到网络中的某一个主机，第二步是用 LLC 帧的地址信息找到该主机中的某一个服务访问点 SAP。这个多进程同时通信的概念是非常重要的。当一个 LLC 子层有多

个服务访问点时，不同的用户使用不同的服务访问点就可以做不同的事情。

2.2.3 媒体接入控制 MAC 子层

在所有计算机系统的设计中，标识系统都是一个核心问题。在标识系统中，地址就是为标识某个系统的一个非常重要的标识符。在 802 标准为局域网上的每一个站规定了一种 48bit 的全局地址。当一个站接入到另一个局域网时，其全局地址并不改变。这个地址就是物理地址或 MAC 地址，也就是通常所说的计算机的硬件地址(hardware address)。在计算机网络中，硬件地址的作用就是用来找到所要进行通信的计算机。网卡从网上每收到一个 MAC 帧就首先检查其硬件地址。如果是发往本站的帧则收下，然后再进行其他的处理。否则就将此帧丢弃，不再进行其它处理。发送的帧包括以下三种帧：

- 单播(unicast)帧，即收到得到帧的 MAC 地址与本站的硬件地址相同；
- 广播(broadcast)帧，即发送给所有站点的帧(全 1 地址)；
- 多播(multicast)帧，即发送给一部分站点的帧。

所有网卡都至少应当能够识别前两种帧，即能够识别单播和广播地址。有的网卡可以用编程的方法识别多播地址。

2.2.4 局域网数据的封装

当应用程序用 TCP 传送数据时，数据被送入协议栈中，然后逐个通过每一层直到被当作一串比特流送入网络。其中每一层对收到的数据都要增加一些首部信息（有时还要增加尾部信息），图 2.4 列出

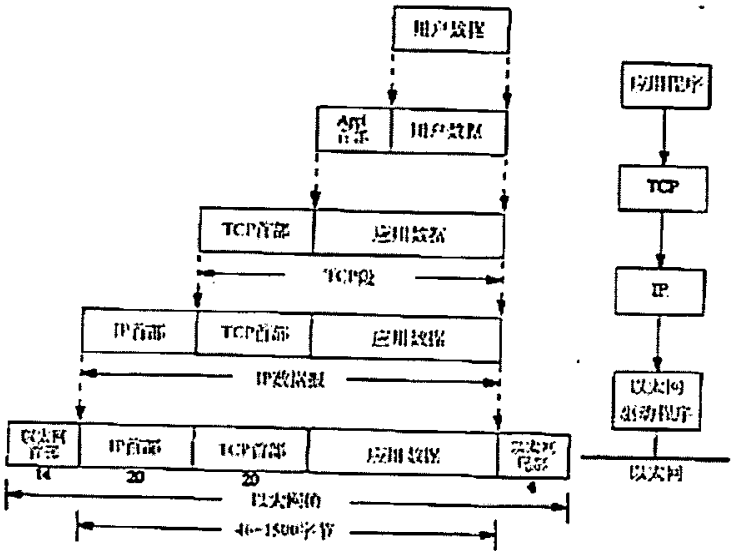

图 2.4　协议数据封装形式图[6]

TCP 传给 IP 的数据单元称作 TCP 报文段或简称为 TCP 段（TCP segment）。IP 传给网络接口层的数据单元称作 IP 数据报(IP datagram)。通过以太网传输的比特流称作帧(Frame)。帧头和帧尾下面所标注的数字是典型以太网帧首部的字节长度。以太网数据帧的物理特性是其长度必须在 46～1500 字节之间。IP 和网络接口层之间传送的数据单元应该是分组（packet）。

第三节　网络互连与网际协议 IP

2.3.1　网际协议 IP[7]

IP 是 TCP/IP 协议族中最为核心的协议。所有的 TCP、UDP、ICMP 及 IGMP 数据都以 IP 数据报格式传输。IP 提供的是不可靠、无连接的数据报传送服务。不可靠(unreliable)的意思是它不能保证 IP 数据报能成功地到达目的地。IP 仅提供最好的传输服务。如果发生某种错误时，如某个路由器暂时用完了缓冲区，IP 有一个简单的错误处理算法：丢弃该数据报，然后发送 ICMP 消息报给信源端。任何要求的可靠性必须由上层来提供（如 TCP）。

无连接(connectionless)这个术语的意思是 IP 并不维护任何关于后续数据

报的状态信息。每个数据报的处理是相互独立的。这也说明，IP 数据报可以不按发送顺序接收。如果一信源向相同的信宿发送两个连续的数据报（先是 A，然后是 B），每个数据报都是独立地进行路由选择，可能选择不同的路线，因此 B 可能在 A 到达之前先到达。网际协议 IP 是 TCP/IP 体系中两个最重要的协议之一。与 IP 协议配套使用的还有三个协议：

- 地址解析协议（Address Resolution Protocol）
- 逆地址解析协议(Reverse Address Resolution Protocol)
- Internet 控制报文协议(Internet Control Message Protocol)

图 2.5　协议框图

图 2.5 为协议框图。在 TCP/IP 体系中，IP 地址是一个重要的概念。所谓 IP 地址就是给每一个连接在 Internet 上的主机分配一个在全世界范围图 2.5 是唯一的 32bit 地址。IP 地址的结构使用户可在 Internet 上很方便地进行寻址，这就是：先按 IP 地址中的网络号 net-id 把网络找到，再按主机号 host-id 把主机找到。所以 IP 地址并不是一个计算机的号，而是指出了连接到某个网络上的某个计算机。IP 地址一共分成五类。常用的是 A 类、B 类、C 类地址都是由两个字段组成，即：网络号字段 net-id。A 类、B 类、C 类地址的网络号字段分别为 1，2 和 3 字节长，在网络号字段的最前面有 1—3bit 的类别比特，其数值分别规定为 0,10,110。主机号字段 host-id。A 类、B 类、C 类地址的主机号字段分别为 3,2 和 1 字节长。图 2.6

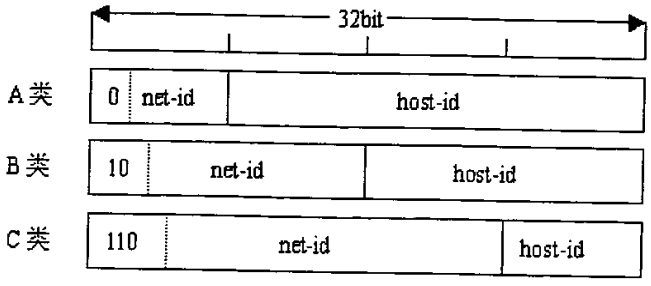

图 2.6　地址数据结构

图 2.6 在以上的说明中，涉及到了两个地址—IP 地址和物理地址。图 2.7 强调了两者的区别。可以看出 IP 地址放在 IP 数据报的首部，而硬件地址则放在 MAC 帧的首部。

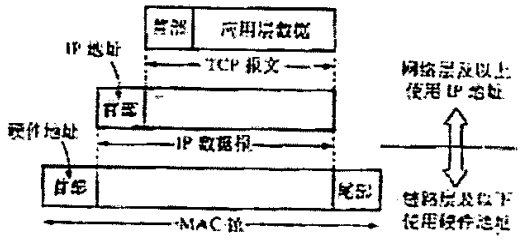

图 2.7　IP 地址和物理地址区别

在网络层及以上使用的是 IP 地址，而链路层及以下使用的是硬件地址。在 IP 层抽象的互连网上，只能看到 IP 数据报，路由器只根据目的站的 IP 地址进行选路。在具体的物理网络的链路层，看到的只是 MAC 帧。IP 数据报被封装在 MAC 帧里面。IP 数据报的格式能够说明 IP 协议都具有什么功能。

在 TCP/IP 的标准中，各种数据格式常常以 32bit 为单位来描述，图 2.8 为 IP 数据报的格式。一个 IP 数据报是由首部和数据两部分组成的。首部的前一部分是固定的 20 个字节，后一部分则是可变长度。IP 数据报中首部的各个固定字段都具有特定的意义，比如版本是指 IP 协议的版本，目前使用的 IP 协议版本为 4。各字段的意义这里不一一列举了。

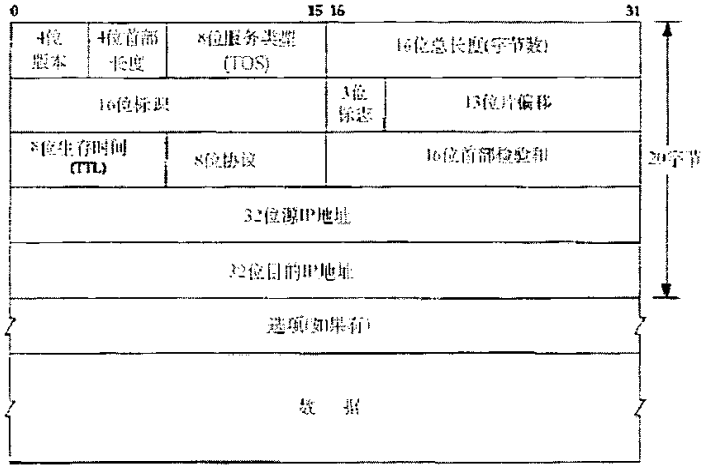

图 2.8　IP 数据报格式图[8]

2.3.2　地址解析协议 ARP

在上文中提到了 IP 地址与物理地址这两种地址，在实际的网络通信中这两种地址要进行转换，这种转换是由地址解析协议(Address Resolution Protocol)来完成的。由于 IP 地址有 32bit，而局域网的物理地址是 48bit，因此它们之间不是一个简单的转换关系。此外，在一个网络上可能会经常有新的计算机加入，或撤走一些计算机。因此应存放一个从 IP 地址到物理地址的转换表，并且能够经常动态更新。地址解析协议 ARP 很好地解决了这些问题。

每个主机都有一个 ARP 高速缓存(ARP cache)，里面有 IP 地址到物理地址的映射表。当主机 A 欲向本局域网上的主机 B 发送一个 IP 数据报时，就先在其 ARP 高速缓存中查看有无主机 B 的 IP 地址。如有，就可查出其对应的物理地址，然后将此物理地址写入 MAC 帧，通过局域网发往此物理地址。当可能查不到主机 B 的 IP 地址，这时主机 A 就自动运行 ARP，按以下步骤找出主机 B 的物理地址：

(1)ARP 进程在本局域网上广播发送一个 ARP 请求分组，上面有主机 B 的 IP 地址；

(2)在本局域网上的所有主机上运行的 ARP 进程都收到此 ARP 请求分组；

(3)主机 B 在 ARP 请求分组中见到自己的 IP 地址，就向主机 A 发送一个 ARP

响应分组，上面写入自己的物理地址；

(4)主机 A 收到主机 B 的 ARP 响应分组后，就在其 ARP 高速缓存中写入主机 B 的 IP 地址到物理地址的映射。

在进行地址转换时，有时还要用到逆地址解析协议 RARP。RARP 使只知道自己物理地址的主机能够知道其 IP 地址。这种主机往往是无盘工作站。

2.3.3 Internet 控制报文协议 ICMP

IP 数据报的传送是不保证不丢失的，为了减少分组的丢失，就要使用 Internet 控制报文协议 ICMP(Internet Control Message Protocol)。ICMP 允许主机或路由器报告差错情况和提供有个异常情况的报告. 但 ICMP 不是高层协议，它仍是 IP 层中的协议。ICMP 报文作为 IP 曾数据报的数据，加上数据报的首部，组成 IP 数据报发送出去。ICMP 报文格式图 2.9 所示。

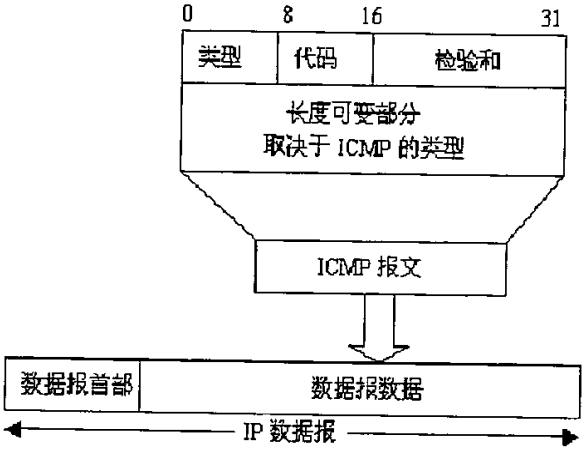

图 2.9 ICMP 报文格式[9]

ICMP 报文的前四个字节是统一的格式，共有三个字段。但后面是长度可变部分，其长度取决于 ICMP 的类型。ICMP 询问报文有如下几种常用的：

ICMP Echo 请求报文是有主机或路由器向一个特定的目的之间发出的询问。在应用层有一个服务叫做 PING，用来测试两个主机之间的连通性。PING 使用了 ICMP Echo 请求与 Echo 回答报文；

ICMP 时间戳请求报文是请某个主机或路由器回答当前的日期和时间；

ICMP 地址掩码请求报文可使主机向子网掩码服务器得到某个接口的地址掩

码。

第四节 运输层协议

图 2.10 中所示，从通信和信息处理的角度看，运输层属于面向通信部分的最高层。但从网络功能或用户功能来划分，则其又属于用户功能的最低层。

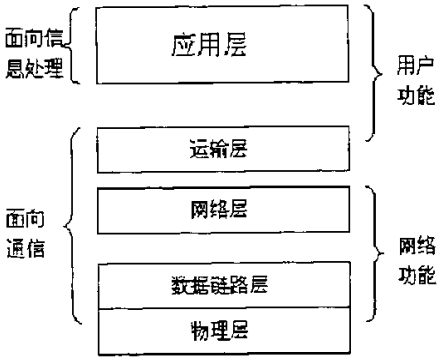

图 2.10 层结构功能图[10]

用户进程希望得到端到端的可靠通信服务。为了能使通信子网的用户得到一个统一的通信服务，就有必要设置一个运输层。它弥补了通信子网提供的服务的差异和不足，使得对两端的网络用户来说，各通信子网都变成透明的。运输层使高层用户看见的就是好像在两个运输层实体之间有一条端到端的、可靠的、全双工通信通路。

UDP 和 TCP 都使用 IP 协议。也就是说，这两个协议在发送数据时，其协议数据单元 PDU 都作为下面 IP 数据报中的数据。在接收数据时，IP 数据报将 IP 首部去掉后，根据上层使用的是什么运输协议，把数据部分交给上层的 UDP 或 TCP。

UDP 在传送数据之前不需要先建立连接。远地主机的运输层在收到 UDP 数据报后，不需要给出任何应答。TCP/IP 体系中的应用服务，如 TFTP 和 NFS 就使用 UDP 这种运算方式。TCP 则提供面向连接的服务。TCP 不提供广播或多播服务。由于 TCP 要提供可靠的运输服务，因此 TCP 就不可避免地增加了许多开销，如应答、流量控制、定时器以及连接管理等。

2.4.1　端口的概念

UDP 和 TCP 都使用了与应用层接口处的端口(port)与上层的应用程序进行通信。应用层的各种进程是通过响应的端口才能与运输实体进行交互。TCP 建立连接时采用客户服务器模式。主动发起连接建立的进程叫做客户(client)，而被动等待连接建立进程叫做服务器(server)。

在运输层与应用层的接口上所设置端口是一个 16bit 的地址，并用端口号进行标识。端口号分为两类。一类是专门分配给一些最常用的应用层程序，叫做熟知端口。“熟知”就表示这些端口号是 TCP/IP 体系确定并公布的，因而是所有用户进程都熟知的。在应用层中的各种不同的服务器进程不断地检测分配给它们的熟知端口，以便发现是否某个客户进程要和它通信。另一类则是一般端口号，用来随时分配给请求通信的客户进程。图 2.11 列举了常用的熟知端口。

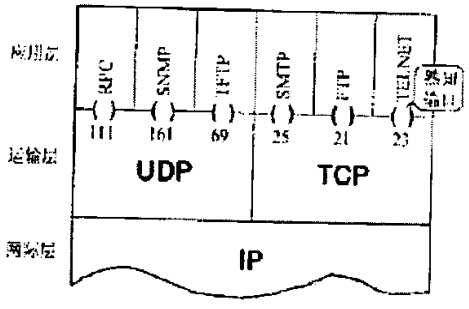

图 2.11　常用端口图[11]

为了在通信时不致发生混乱，就必须把端口号和主机 IP 地址结合在一起使用。当两个主机使用同一个端口号时，只要查一下 IP 地址就可知道是哪一个主机的数据。因此，TCP 使用“连接”(而不仅仅是“端口”)作为最基本的抽象。一个连接由他的两个端点来标识。这样的端点就叫插口(socket)，或套接字。插口包括 IP 地址(32bit)和端口号(16bit)，共 48bit。在整个 Internet 中，在运输层通信的一对插口必须是唯一的。

2.4.2　用户数据报协议

UDP 用户数据报协议 UDP 只在 IP 的数据报服务之上增加很少一点的功能，这就是端口的功能。用户数据报 UDP 有两个字段：数据字段和首部字段。首部

字段很简单，只有 8 个字节，由 4 个字段组成，每个字段都是两个字节。各字段意义如下：

- 源端口字段 源端口号
- 目的端口字段 目的端口号
- 长度字段 UDP 数据报的长度
- 检验和字段 防止 UDP 数据报在传输中出错，如图 2.12 所示

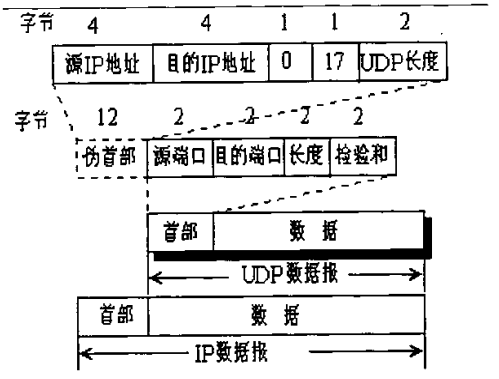

图 2.12 UDP 用户数据报字段

UDP 数据报首部中检验和的计算方法有些特殊。在计算检验和时在 UDP 数据报之前要增加 12 个字节的伪首部。所谓“伪首部”是因为这种首部并不是 UDP 数据报真正的首部。只是在计算检验和时，临时和 UDP 数据报连接在一起，得到一个过渡的 UDP 数据报。检验和就是按照这个过渡的 UDP 数据报来计算的。伪首部既不向下传送，也不向上递交。伪首部的第 3 字段全为零，第 4 字段是 IP 首部的协议字段的值。对于 UDP，此协议字段值为 17。第 5 字段是 UDP 数据报的长度。

2.4.3 传输控制协议 TCP[12]

尽管 TCP 和 UDP 都使用相同的网络层 IP，TCP 却向应用层提供与 UDP 完全不同的服务。TCP 提供一种面向连接的、可靠的字节流服务。面向连接意味着两个使用 TCP 的应用(通常是一个客户和一个服务器)在彼此交换数据之前必须先建立一个 TCP 连接。这一过程与打电话很相似，先拨号振铃，等待对方摘机说“喂”，然后才说明是谁。在一个 TCP 连接中，仅有两方进行彼此通信。

TCP 通过下列方式来提供可靠性：

(1)应用数据被分割成 TCP 认为最适合发送的数据块。这和 UDP 完全不同，应用程序产生的数据报长度将保持不变。由 TCP 传递给 IP 的信息单位称为报文段或段(segment)；

(2)当 TCP 发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段；

(3)当 TCP 收到发自 TCP 连接另一端的数据，它将发送一个确认。这个确认不是立即发送，通常将推迟几分之一秒；

(3)TCP 将保持它首部和数据的检验和。这是一个端到端的检验和，目的是检测数据在传输过程中的任何变化。如果收到段的检验和有差错，TCP 将丢弃这个报文段和不确认收到此报文段(希望发端超时并重发)；

(4)既然 TCP 报文段作为 IP 数据报来传输，而 IP 数据报的到达可能会失序，因此 TCP 报文段的到达也可能会失序。如果必要，TCP 将对收到的数据进行重新排序，将收到的数据以正确的顺序交给应用层；

(5)既然 IP 数据报会发生重复，TCP 的接收端必须丢弃重复的数据；

(6)TCP 还能提供流量控制。TCP 连接的每一方都有固定大小的缓冲空间，TCP 的接收端只允许另一端发送接收端缓冲区所能接纳的数据。这将防止较快主机致使较慢主机的缓冲区溢出。

TCP 为应用层提供全双工服务。这意味数据能在两个方向上独立地进行传输。因此，连接的每一端必须保持每个方向上的传输数据序号。TCP 可以表述为一个没有选择确认或否认的滑动窗口协议。TCP 缺少选择确认是因为 TCP 首部中的确认序号表示发方已成功收到字节，但还不包含确认序号所指的字节。当前还无法对数据流中选定的部分进行确认。

TCP 提供了一种可靠的面向连接的字节流运输层服务。TCP 将用户数据打包构成报文段；它发送数据后启动一个定时器；另一端对收到的数据进行确认，对失序的数据重新排序，丢弃重复数据；TCP 提供端到端的流量控制，并计算和验证一个强制性的端到端检验和。许多流行的应用程序如 Telnet、Rlogin、FTP 和 SMTP 都使用 TCP。

第五节　应用层协议

应用层是原理体系结构中的最高层，在应用层之上不存在其他层，每个应

用层协议都是为了解决某一类应用问题，而问题的解决又往往是通过位于不同主机的多个进程之间的通信和协同工作来完成的。客户服务器模式所描述的是进程之间服务和被服务的关系。当 A 进程需要 B 进程的服务时就主动呼叫 B 进程，在这种情况下，A 是客户而 B 是服务器。因此最主要的特征就是：客户是主叫方，服务器是被叫方。客户与服务器的通信关系一旦建立，通信就是双向的，客户和服务器都可发送和接收信息。对于大多数的应用程序都是使用 TCP/IP 协议进行通信。

在应用层的协议包括文件传送协议 FTP(File Transfer Protocol)，简单文件传送协议 TFTP(Trivial File Transfer Protocol)，远程登录 TELNET，电子邮件系统以及万维网 WWW 等多种为人们说熟知的协议。

第三章 μC/OS-Ⅱ嵌入式操作系统

第一节 实时操作系统的概念

提倡在计算机嵌入式应用中使用实时操作系统 RTOS(Real Time Operating System)，是因为 RTOS 将应用分解成多任务，简化了应用系统软件的设计；RTOS 使控制系统的实时性得到保证，可以接近理论上能达到的最好水平；良好的多任务设计，有助于提高系统的稳定性与可靠性。[13]

μC/OS-Ⅱ作为嵌入式操作系统的一只新军，是近几年才为人们所认识的。它是一个完整的、可移植、固化、裁剪的占先实时多任务内核。μC/OS-Ⅱ是用 ANSI 的 C 语言编写的，包含一小部分语言代码，使之可供不同构架的微处理器使用。世界上已有在各个领域中使用的μC/OS，这些领域包括：照相机行业、航空业、医疗器械和工业机器人等。

实时操作系统的特点是，如果逻辑和时序出现偏差，将会引起严重的后果。有两种类型的实时系统：软实时系统和硬实时系统。

在软实时系统中，系统的宗旨是使各个任务尽快的运行，而不要求限定某一任务在多长时间内完成；

在硬实时系统中，各任务不仅须执行无误，而且要做到准时。大多数实时系统是两者的结合。每个任务都是一个无限的循环，都处在以下 5 种状态之一：休眠态、就绪态、运行态、挂起态及被中断态。

第二节 μC/OS-II 操作系统

μC/OS-II 内核是可剥夺型实时内核，它的性能并不亚于商业级 RTOS，不过商业级软件通常都提供了包括编程、调试和下载等功能的完整的开发包，μC/OS-II 只是一个提供源代码的操作系统。它的开发环境目前在 PC 平台上可以采用 Borland C/C++ 3.11 或 4.5 版本，下载则需要用到与存储介质配套的工具如编程器。μC/OS-II 还可以在 DOS 下运行，即系统先由 DOS 启动，然后运行嵌入了μC/OS-II 的应用程序。[14]

μC/OS-II 主要是针对嵌入式应用，用户可以对它进行移植、裁剪和固化。为了减小操作系统内核所占体积，用户可以根据实际使用情况来决定编译那些部分，内核的目标代码可以裁减到 1.5k。

μC/OS-II 是一个多任务实时操作系统，任务是系统执行和调度的基本单位。这里的任务与其它操作系统中描述的进程基本类似。系统可以管理约 60 个用户任务，每个任务都有自己的栈空间，用户任务越多，需要到 RAM 就越大。

任务设计为五种状态:睡眠状态、就绪状态、等待状态、运行状态和中断服务状态。各状态之间的关系和转换途径如图 3-1 所示。

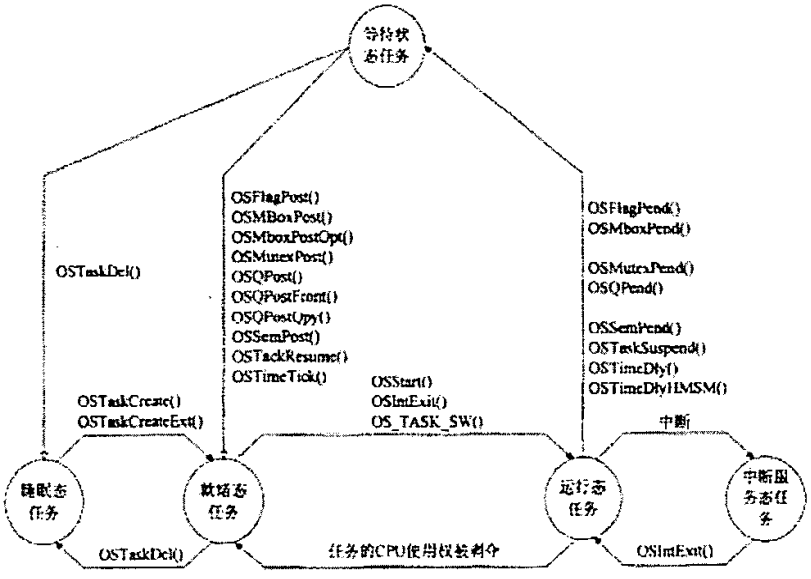

图 3-1　各状态之间关系和转换图[15]

任务的调度采用可剥夺式优先级调度。任务的优先级在任务创建时指定，当任务运行时还可以通过 OSTaskChangePrio()函数进行修改。

μC/OS-II 总是运行进入就绪态任务中优先级最高的任务。就绪态的任务都保存在 OSRdyTb()结构中，任务优先级的判定由调度函数 OSSched()确定。当调度函数发现有更重要的任务需要运行，就通过调用函数 OSASKSW()实现任务切换。

多任务系统中不可避免会出现任务同步问题。μC/OS-II 提供了信号量和互斥型信号量为任务同步提供支持。信号量和互斥量都提供了创建、删除、获取、释放和查询等操作。

为方便多任务之间进行通信，μC/OS-II 提供了消息邮箱和消息队列。消息邮箱和消息队列都提供了创建、删除、等待消息、发送消息、获取消息和查询状态等操作;其中发送消息都可以发送广播消息。消息队列还提供了清空消息方法。

内存管理是操作系统的重要组成部分。嵌入式操作系统由于其硬件资源的特殊性，通常不能频繁使用 malloc()，free()，因为多次调用这两个函数，会把原来很大的一块连续内存逐渐分割成许多非常小且彼此不相邻的内存块，即内存碎片，而嵌入式操作系统通常也不会设计复杂的内存回收办法避免出现内存碎片。为了简化内存管理，在μC/OS-II 中采用把连续的大块内存按分区来管理的方式，每个分区包含整数个大小相同的内存块。

当应用程序需要分配内存时，就从最接近的内存分区上取下一块:使用完后再整块归还，这样就避免了内存碎片的出现，也简化了内存分配和回收过程。

μC/OS-II 还提供了时间管理、事件控制块和事件标志组等机制方便用户开发使用。

3.2.1 μC/OS-Ⅱ特点[16]

(1) μC/OS-Ⅱ是一个占先式的内核，即已经准备就绪的高优先级任务可以剥夺正在运行的低优先级任务的 CPU 使用权。这个特点使得它的实时性比非占先式的内核要好。通常都是在中断服务程序中使高优先级任务进入就绪态(例如发信号)，这样退出中断服务程序后，将进行任务切换，高优先级任务将被执行。如果使用μC/OS-Ⅱ的话，只要把数据处理程序的优先级设定的高一些，并在中断服务程序中使它进入就绪态，中断结束后数据处理程序就会被立即执行。这样可以把中断响应时间限制在一定的范围内。

(2) μC/OS-Ⅱ不支持时间片轮转法。它是一个基于优先级的实时操作系统。每一个任务的优先级必须不同(分析它的源码会发现，μC/OS-Ⅱ把任务的优先级当作任务的标识来使用，如果优先级相同，任务将无法区分)。进入就绪态的优先级最高的任务首先得到 CPU 的使用权，只有等它交出 CPU 的使用权后，其他任务才可以被执行。所以它只能说是多任务，不能说是多进程，至少不是通常熟悉的那种多进程，因此对于实时处理非常重要的任务是不合适使用的。

(3) μC/OS-Ⅱ对共享资源提供了保护机制。μC/OS-Ⅱ是一个支持多任务

的操作系统。可以把一个完整的程序划分成几个任务，不同的任务执行不同的功能。对于共享资源(比如串口)，μC/OS-Ⅱ也提供了很好的解决办法，一般情况下使用的是信号量方法。创建一个信号量并对它进行初始化，当一个任务需要使用一个共享资源时，它必须先申请得到这个信号量。在这个过程中即使有优先权更高的任务进入了就绪态，因为无法得到信号量，也不能使用该资源。在 uC/OS-Ⅱ中称为优先级反转,所以在使用μC/OS-Ⅱ时，必须对所开发的系统了解清楚才能选择对于某种共享资源是否使用信号量。

3.2.2 μC/OS-Ⅱ在单片机中的应用[17]

(1) 在单片机系统中嵌入μC/OS-Ⅱ将增强系统的可靠性，并使得调试程序变得简单起来。如果在系统中嵌入μC/OS-Ⅱ，可以把整个程序分成许多任务，每个任务相对独立。然后在每个任务中设置超时函数，时间用完以后，任务必须交出 CPU 的使用权。即使一个任务发生问题，也不会影响其它任务的运行。这样既提高了系统的可靠性，同时也使得调试程序变得容易。。

(2) 在单片机系统中嵌入μC/OS-Ⅱ将增加系统的开销，这在许多书籍和资料中都提到过。现在所使用的 51 系列单片机，其片内都带有一定的 RAM 和 ROM。对于一些简单的程序，如果采用传统的编程方法，已经不需要外扩存储器了。由于μC/OS-Ⅱ是可裁减的操作系统，其所需要的 RAM 大小就依赖于我们对操作系统一些功能的选择。嵌入μC/OS-Ⅱ以后总的 RAM 需求可以由如下表达式得出：

RAM 总需求=应用程序的 RAM 需求+内核数据区的 RAM 需求+(任务栈需求+最大中断嵌套栈需求)×任务数

μC/OS-Ⅱ可以对每个任务分别定义堆栈空间的大小，可根据任务的实际需求来进行栈空间的分配。在 RAM 容量有限的情况下，要注意对大型数组、数据结构和函数的使用，，函数的形参也要推入堆栈。

(3) 和其它一些著名的嵌入式操作系统不同μC/OS-Ⅱ在单片机系统中的启动过程比较简单。μC/OS-Ⅱ的内核是和应用程序放在一起编译成一个文件的，只需要把这个文件转换成 HEX 格式，写入 ROM 中就可以了。上电后，可以象普通的单片机程序一样运行。

3.2.3 临界段管理[18]

代码的临界段指处理是不可分割的代码。一旦这部分代码开始执行，则不允许任何中断打入。为确保临界段代码的执行，在进入临界段之前要关中断，而临界段代码执行完以后要立即开中断。微处理器一般都有关中断/开中断指令，用户使用的C语言编译器必须有某种机制能够在C中直接实现关中断/开中断地操作。μC/OS-Ⅱ定义两个宏(macros)来关中断和开中断，以便避开不同C编译器厂商选择不同的方法来处理关中断和开中断。μC/OS-Ⅱ中的这两个宏调用分别是：OS_ENTER_CRITICAL()和OS_EXIT_CRITICAL()。因为这两个宏的定义取决于所用的微处理器，故在文件OS_CPU.H中可以找到相应宏定义。每种微处理器都有自己的OS_CPU.H文件。

3.2.4 任务调度机制[19]

内核的核心任务是任务调度机制，对μC/OS进行分析，应从任务调度开始。在μC/OS 中，一个任务通常是一个无限循环，程序具有如下的结构，后面将阐述为什么会有这种结构。从下面的结构可以看出一个任务就像其他C函数一样；而且，既然任务是一个无限循环，它一定不会返回任何的数据，所以返回类型应该定义为void。

程序：

```
void mytask(void *pdata)
  {
   for (;;)
    {
     do something;
     waiting;
     do something;
    }
  }
```

任务被创建后，其状态用8位字节变量OSTCBStat表示，目前只用了低四位。如果某位置为1，表示任务正在等待该位表示的事件；可以复合使用这些标志，表示任务在同时等待多个事件的发生；如果所有位均为0，表示任务处于就绪状态，一旦符合条件(优先级最高)，即可投入运行。

μC/OS-Ⅱ可以管理多达 64 个任务，但目前版本的μC/OS-Ⅱ有两个任务已经被系统占用了。保留了优先级为 0、1、2、3、OS_LOWEST_PRIO-3、OS_LOWEST_PRIO-2，OS_LOWEST_PRIO-1 以及 OS_LOWEST_PRIO 这 8 个任务优先级以被将来使用。OS_LOWEST_PRIO 是作为定义的常数在 OS_CFG.H 文件中用定义常数语句#define constant 定义的。因此用户可以有多达 56 个应用任务。必须给每个任务赋以不同的优先级，优先级可以从 0 到 OS_LOWEST_PRIO-2。优先级号越低，任务的优先级越高。μC/OS-Ⅱ总是运行进入就绪态的优先级最高的任务。任务状态如图 3.2。[20]

睡眠态（DORMANT）指任务驻留在程序空间之中，还没有交给μC/OS-Ⅱ管理。把任务交给μC/OS-Ⅱ是通过调用下述两个函数之一：OSTaskCreate()或 OSTaskCreateExt()。

当任务一旦建立，这个任务就进入就绪态准备运行。如果一个任务是被另一个任务建立的，而这个任务的优先级高于建立它的那个任务，则这个刚刚建立的任务将立即得到 CPU 的控制权。一个任务可以通过调用 OSTaskDel()返回到睡眠态，或通过调用该函数让另一个任务进入睡眠态。

调用 OSStart()可以启动多任务。OSStart()函数运行进入就绪态的优先级最高的任务。就绪的任务只有当所有优先级高于这个任务的任务转为等待状态，或者是被删除了，才能进入运行态。正在运行的任务可以通过调用两个函数之一将自身延迟一段时间，这两个函数是 OSTimeDly()或 OSTimeDlyHMSM()。这个任务于是进入等待状态，等待这段时间过去，下一个优先级最高的、并进入了就绪态的任务立刻被赋予了 CPU 的控制权。

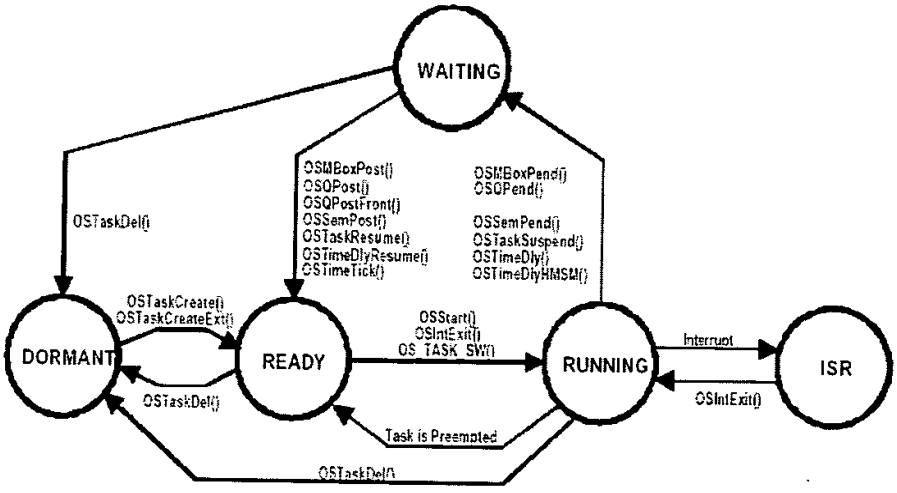

图 3.2 任务状态图[21]

正在运行的任务是可以被中断的，除非该任务将中断关了，或者μC/OS-II将中断关了。被中断了的任务就进入了中断服务态（ISR）。响应中断时，正在执行的任务被挂起，中断服务子程序控制了 CPU 的使用权。中断服务子程序可能会报告一个或多个事件的发生，而使一个或多个任务进入就绪态。在这种情况下，从中断服务子程序返回之前，μC/OS-II要判定，被中断的任务是否还是就绪态任务中优先级最高的。如果中断服务子程序使一个优先级更高的任务进入了就绪态，则新进入就绪态的这个优先级更高的任务将得以运行，否则原来被中断了的任务才能继续运行。

3.2.5 时间管理[22]

μC/OS-II(其它内核也一样)要求用户提供定时中断来实现延时与超时控制等功能。这个定时中断叫做时钟节拍，它应该每秒发生 10 至 100 次。时钟节拍的实际频率是由用户的应用程序决定的。时钟节拍的频率越高，系统的负荷就越重。[23]

(1)任务延时

μC/OS-II提供了这样一个系统服务：申请该服务的任务可以延时一段时间，这段时间的长短是用时钟节拍的数目来确定的。实现这个系统服务的函数

叫做 OSTimeDly()。调用该函数会使μC/OS-Ⅱ进行一次任务调度，并且执行下一个优先级最高的就绪态任务。任务调用 OSTimeDly()后，一旦规定的时间期满或者有其它的任务通过调用 OSTimeDlyResume()取消了延时，它就会马上进入就绪状态。只有当该任务在所有就绪任务中具有最高的优先级时，它才会立即运行。OSTimeDly()虽然是一个非常有用的函数，但用户的应用程序需要知道延时时间对应的时钟节拍的数目。OSTimeDlyHMSM()是使用户可以按小时(H)、分(M)、秒(S)和毫秒(m)来定义时间了，这样会显得更自然些。与 OSTimeDly()一样，调用 OSTimeDlyHMSM()函数也会使μC/OS-Ⅱ进行一次任务调度，并且执行下一个优先级最高的就绪态任务。任务调用 OSTimeDlyHMSM()后，一旦规定的时间期满或者有其它的任务通过调用 OSTimeDlyResume()取消了延时，它就会马上处于就绪态。同样，只有当该任务在所有就绪态任务中具有最高的优先级时，它才会立即运行处在延时期的任务结束延时。

(2)系统时间

无论时钟节拍何时发生，μC/OS-Ⅱ都会将一个 32 位的计数器加 1。这个计数器在用户调用 OSStart()初始化多任和 4,294,967,295 个节拍执行完一遍的时候从 0 开始计数。在时钟节拍的频率等于 100Hz 的时候，这个 32 位的计数器每隔 497 天就重新开始计数。用户可以通过调用 OSTimeGet()来获得该计数器的当前值。也可以通过调用 OSTimeSet()来改变该计数器的值。

3.2.6　任务之间的通信与同步[23]

在μC/OS-Ⅱ中，有多种方法可以保护任务之间的共享数据和提供任务之间的通讯。除了共享数据以外还可以有信号量、邮箱和消息队列。一个任务或者中断服务子程序可以通过事件控制块 ECB（Event Control Blocks）来向另外的任务发信号。

多个任务可以同时等待同一个事件的发生。在这种情况下，当该事件发生后，所有等待该事件的任务中，优先级最高的任务得到了该事件并进入就绪状态，准备执行。上面讲到的事件，可以是信号量、邮箱或者消息队列等。当事件控制块是一个信号量时，任务可以等待它，也可以给它发送消息。

(1)事件控制块

μC/OS-Ⅱ通过 uCOS_II.H 中定义的 OS_EVENT 数据结构来维护一个事件控

制块的所有信息，也就是事件控制块 ECB。该结构中除了包含了事件本身的定义，如用于信号量的计数器，用于指向邮箱的指针，以及指向消息队列的指针数组等，还定义了等待该事件的所有任务的列表。每个信号量、消息邮箱及消息队列都应分配一个事件控制块 ECB。

(2)信号量管理

μC/OS-Ⅱ中的信号量由两部分组成：一个是信号量的计数值，它是一个 16 位的无符号整数（0 到 65,535 之间）；另一个是由等待该信号量的任务组成的等待任务表。用户要在 OS_CFG.H 中将 OS_SEM_EN 开关量常数置成 1，这样 μC/OS-II 才能支持信号量。

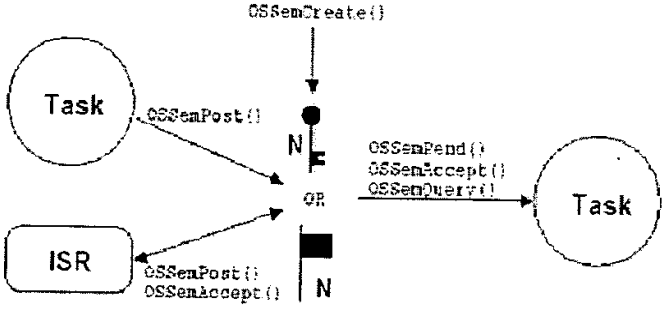

图 3.3　任务、中断服务子程序和信号量关系图

μC/OS-Ⅱ提供了 5 对信号量进行操作的函数。它们是：建立一个信号量 OSSemCreate()，等待一个信号量 OSSemPend()，发送一个信号量 OSSemPost()，无等待的请求一个信号量 OSSemAccept()和查询一个信号量 OSSemQuery()函数。图 3.3 说明了任务、中断服务子程序和信号量之间的关系。图中用钥匙或者旗帜的符号来表示信号量：如果信号量用于对共享资源的访问，那么信号量就用钥匙符号。符号旁边的数字 N 代表可用资源数。对于二值信号量，该值就是 1；如果信号量用于表示某事件的发生，那么就用旗帜符号。这时的数字 N 代表事件已经发生的次数。从图中可以看出 OSSemPost()函数可以由任务或者中断服务子程序调用，而 OSSemPend()和 OSSemQuery()函数只能有任务程序调用。

(3)邮箱

邮箱是μC/OS-Ⅱ中另一种通讯机制，它可以使一个任务或者中断服务子程序向另一个任务发送一个指针型的变量。该指针指向一个包含了特定“消息”的数据结构。为了在μC/OS-Ⅱ中使用邮箱，必须将 OS_CFG.H 中的 OS_MBOX_EN 常数置为 1。

使用邮箱之前，必须先建立该邮箱。该操作可以通过调用 OSMboxCreate()函数来完成，并且要指定指针的初始值。一般情况下，这个初始值是 NULL，但也可以初始化一个邮箱，使其在最开始就包含一条消息。如果使用邮箱的目的是用来通知一个事件的发生（发送一条消息），那么就要初始化该邮箱为 NULL，因为在开始时，事件还没有发生。如果用户用邮箱来共享某些资源，那么就要初始化该邮箱为一个非 NULL 的指针。在这种情况下，邮箱被当成一个二值信号量使用。

μC/OS-II 提供了 5 种对邮箱的操作

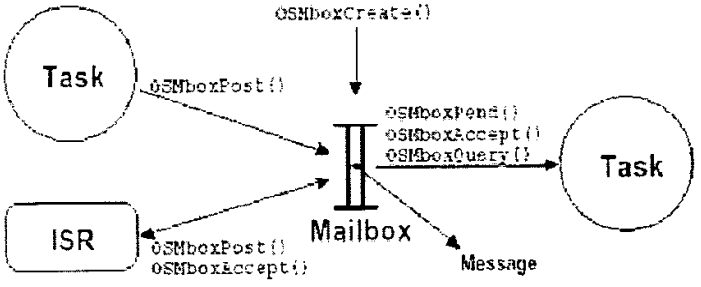

图 3.4　任务、中断服务子程序和邮箱关系图

建立一个邮箱 OSMboxCreate()，等待一个邮箱中的消息 OSMboxPend()，发送一个消息到邮箱中 OSMboxPost()，无等待地从邮箱中得到一个消息 OSMboxAccept()和查询一个邮箱的状态 OSMboxQuery()函数。图 3.4 描述了任务、中断服务子程序和邮箱之间的关系，这里用符号“I”表示邮箱。邮箱包含的内容是一个指向一条消息的指针。一个邮箱只能包含一个这样的指针（邮箱为满时），或者一个指向 NULL 的指针（邮箱为空时）。从图可以看出，任务或者中断服务子程序可以调用函数 OSMboxPost()，但是只有任务可以调用函数 OSMboxPend()和 OSMboxQuery()。

(4)消息队列

消息队列是μC/OS-II 中另一种通讯机制，它可以使一个任务或者中断服务子程序向另一个任务发送以指针方式定义的变量。因具体的应用有所不同，每个指针指向的数据结构变量也有所不同。为了使用μC/OS-II 的消息队列功能，需要在 OS_CFG.H 文件中，将 OS_Q_EN 常数设置为 1，并且通过常数 OS_MAX_QS 来决定μC/OS-II 支持的最多消息队列数。在使用一个消息队列之前，必须先建立该消息队列。这可以通过调用 OSQCreate()函数，并定义消息队列中的单元数

（消息数）来完成。

μC/OS-II 提供了 7 个对消息队列进行操作的函数：建立一个消息队列 OSQCreate()，等待一个消息队列中的消息 OSQPend()，向消息队列发送一个消息(FIFO)OSQPost()，向消息队列发送一个消息(LIFO)OSQPostFront()，无等待地从消息队列中取得消息 OSQAccept()，清空一个消息队列 OSQFlush()和查询一个消息队列的状况 OSQQuery()函数。

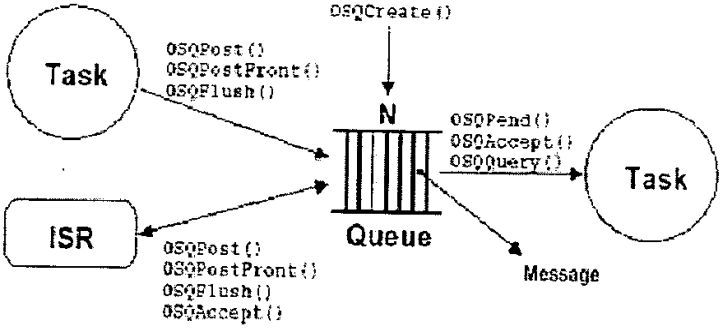

图 3.5 任务、中断服务子程序和消息队列关系图[24]

图 3.5 是任务、中断服务子程序和消息队列之间的关系。其中，消息队列的符号很像多个邮箱。实际上，可将消息队列看作多个邮箱组成的数组，只是它们共用一个等待任务列表。每个指针所指向的数据结构是由具体的应用程序决定的。N 代表了消息队列中的总单元数。当调用 OSQPend()或者 OSQAccept()之前，调用 N 次 OSQPost()或者 OSQPostFront()就会把消息队列填满。从图中可以看出，一个任务或者中断服务子程序可以调用 OSQPost()，OSQPostFront()，OSQFlush()或者 OSQAccept()函数。但是，只有任务可以调用 OSQPend()和 OSQQuery()函数。

3.2.7 内存管理[25]

在 ANSI C 中可以用 malloc()和 free()两个函数动态地分配内存和释放内存。但是，在嵌入式实时操作系统中，多次这样做会把原来很大的一块连续内存区域，逐渐地分割成许多非常小而且彼此又不相邻的内存区域，也就是内存碎片。由于这些碎片的大量存在，使得程序到后来连非常小的内存也分配不到。

在μC/OS-II 中，操作系统把连续的大块内存按分区来管理。每个分区中包含有整数个大小相同的内存块。利用这种机制，μC/OS-II 对 malloc()和 free()

函数进行了改进，使得它们可以分配和释放固定大小的内存块。这样一来，malloc()和 free()函数的执行时间也是固定的了。

第三节　μC/OS-Ⅱ在微处理器上的移植

所谓移植，就是使一个实时内核能在某个微处理器或微控制器上运行。为了方便移植，大部分的μC/OS-Ⅱ代码是用 C 语言写的；但仍需要用 C 和汇编语言写一些与处理器相关的代码，这是因为μC/OS-Ⅱ在读写处理器寄存器时只能通过汇编语言来实现。由于μC/OS-Ⅱ在设计时就已经充分考虑了可移植性，所以μC/OSⅡ的移植相对来说是比较容易的。要使μC/OS-Ⅱ正常运行，处理器必须满足以下要求：

- 处理器的 C 编译器能产生可重入代码；
- 用 C 语言就可以打开和关闭中断；
- 处理器支持中断，并且能产生定时中断(通常在 10 至 100Hz 之间)；
- 处理器支持能够容纳一定量数据(可能是几千字节)的硬件堆栈；
- 处理器有将堆栈指针和其它 CPU 寄存器读出和存储到堆栈或内存中的指令。

移植工作包括以下几个内容：[26]

- 用#define 设置一个常量的值(OS_CPU.H)
- 声明 10 个数据类型(OS_CPU.H)
- 用#define 声明三个宏(OS_CPU.H)
- 用 C 语言编写六个简单的函数(OS_CPU_C.C)
- 编写四个汇编语言函数(OS_CPU_A.ASM)

3.3.1　移植的过程：[27]

(1)改写最简单的 OS_CPU.H 数据类型的设定按照 C51 的定义来设定。BOOLEAN 定义成 unsigned char 类型，因为 bit 类型为 C51 特有，不能用在结构体里。宏定义 EA=0 关中断；EA=1 开中断。这样定义即减少了程序行数，又避免了退出临界区后关中断造成的死机。MCS-51 堆栈从下往上增长(1=向下，0=向上)，OS_STK_GROWTH 定义为 0 #define OS_TASK_SW() OSCtxSw()因为 MCS-51

没有软中断指令，所以用程序调用代替。两者的堆栈格式相同，RETI 指令复位中断系统，RET 则没有。实践表明，对于 MCS-51，用子程序调用入栈，用中断返回指令 RETI 出栈是没有问题的，反之中断入栈 RET 出栈则不行。总之，对于入栈，子程序调用与中断调用效果是一样的，可以混用。在没有中断发生的情况下复位中断系统也不会影响系统正常运行。

(2)改写 OS_CPU_C.C

OSTaskStkInit()

OSTaskCreateHook()

OSTaskDelHook()

OSTaskSwHook()

OSTaskStatHook()

OSTimeTickHook()

唯一必要的函数是 OSTaskStkInit()，其它五个函数必须得声明但没必要包含代码。OSTaskCreate()和 OSTaskCreateExt()通过调用 OSTaskStkInt()来初始化任务的堆栈结构，因此，堆栈看起来就像刚发生过中断并将所有的寄存器保存到堆栈中的情形一样。

在用户建立任务的时候，用户会传递任务的地址，pdata 指针，任务的堆栈栈顶和任务的优先级给 OSTaskCreate()和 OSTaskCreateExt()。虽然 OSTaskCreateExt()还要求有其它的参数，但这些参数在讨论 OSTaskStkInt()的时候是无关紧要的。为了正确初始化堆栈结构，OSTaskStkInt()只要求刚才提到的前三个参数和一个附加的选项，这个选项只能在 OSTaskCreateExt()中得到。设计的堆栈结构如图 3.6 所示：

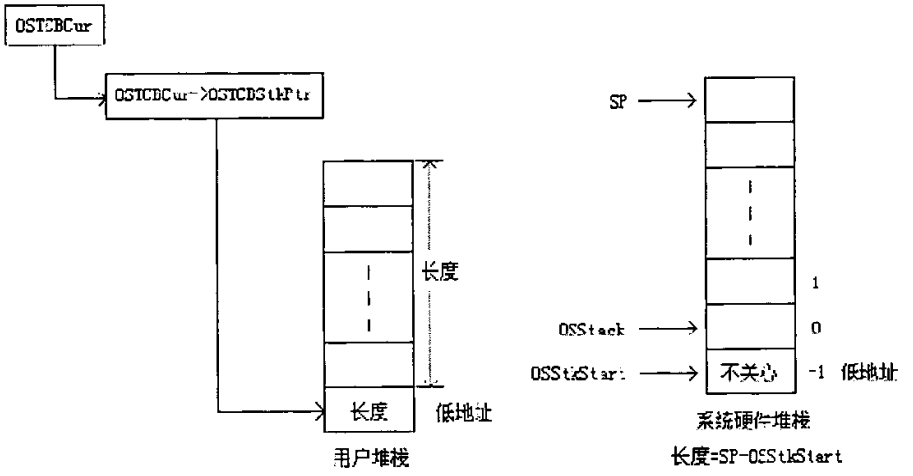

图 3.6　堆栈结构图[28]

TCB 结构体中 OSTCBStkPtr 总是指向用户堆栈最低地址，该地址空间内存放用户堆栈长度，其上空间存放系统堆栈映像，即：用户堆栈空间大小=系统堆栈空间大小+1。SP 总是先加 1 再存数据，因此，SP 初始时指向系统堆栈起始地址(OSStack)减 1 处(OSStkStart)。很明显系统堆栈存储空间大小=SP-OSStkStart。

任务切换时，先保存当前任务堆栈内容。方法是：用 SP-OSStkStart 得出保存字节数，将其写入用户堆栈最低地址内，以用户堆栈最低地址为起址，以 OSStkStart 为系统堆栈起址，由系统栈向用户栈拷贝数据，循环 SP-OSStkStart 次，每次拷贝前先将各自栈指针增 1。其次，恢复最高优先级任务系统堆栈。方法是：获得最高优先级任务用户堆栈最低地址，从中取出“长度”，以最高优先级任务用户堆栈最低地址为起址，以 OSStkStart 为系统堆栈起址，由用户栈向系统栈拷贝数据，循环“长度”数值指示的次数，每次拷贝前先将各自栈指针增 1。

用户堆栈初始化时从下向上依次保存：用户堆栈长度（15），PCL，PCH，PSW，ACC，B，DPL，DPH，R0，R1，R2，R3，R4，R5，R6，R7。不保存 SP，任务切换时根据用户堆栈长度计算得出。OSTaskStkInit 函数总是返回用户栈最低地址。操作系统 tick 时钟使用了 51 单片机的 T0 定时器。

最后还有几点必须注意的事项。原则上不用修改与处理器无关的代码，但是由于 KEIL 编译器的特殊性，这些代码仍要多处改动。因为 KEIL 缺省情况下

编译的代码不可重入，而多任务系统要求并发操作导致重入，所以要在每个 C 函数及其声明后标注 reentrant 关键字。OSTCBCur、OSTCBHighRdy、OSRunning、OSPrioCur、OSPrioHighRdy 这几个变量在汇编程序中用到了，为了使用 Ri 访问而不用 DPTR，应该用 KEIL 扩展关键字 IDATA 将它们定义在内部 RAM 中。

(3)重写 OS_CPU_A.ASM

A51 宏汇编的大致结构如下：

NAME 模块名 ; 与文件名无关

定义重定位段 ; 必须按照 C51 格式定义，汇编遵守 C51 规范。

段名格式为： PR 函数名 模块名

;声明引用全局变量和外部子程序 注意关键字为“EXTRN”没有‘E’

全局变量名直接引用

无参数/无寄存器参数函数 FUNC

带寄存器参数函数 _FUNC

重入函数 _FUNC

;分配堆栈空间

只关心大小，堆栈起点由 keil 决定，通过标号可以获得 keil 分配的 SP 起点。切莫自己分配堆栈起点，只要用 DS 通知 KEIL 预留堆栈空间即可。STACK 段名与 STARTUP.A51 中的段名相同，这意味着 KEIL 在 LINK 时将把两个同名段拼在一起，预留 40H 个字节，STARTUP.A51 预留了 1 个字节，LINK 完成后堆栈段总长为 41H。查看 yy.m51 知 KEIL 将堆栈起点定在 21H，长度 41H，处于内部 RAM 中。

;定义宏

宏名 MACRO 实体 ENDM

;子程序

OSStartHighRdy

OSCtxSw

OSIntCtxSw

OSTickISR

SerialISR

END ; 声明汇编源文件结束

一般指针占 3 字节。+0 类型+1 高 8 位数据+2 低 8 位数据

低位地址存高 8 位值，高位地址存低 8 位值。例如 0x1234，基址+0:0x12 基址+1:0x34

3.3.2 测试移植代码

当在处理器上做完μC/OS-Ⅱ的移植后，紧接着的工作就是验证移植的μC/OS-Ⅱ是否正常工作，而这是移植中最复杂的一步。应该首先不加任何应用代码来测试移植的μC/OS-Ⅱ，也就是说，应该首先测试内核自身的运行情况。有两个原因促使这样做：首先，用户不希望将事情复杂化；其次，如果有些部分没有正常工作，可以明白是移植本身的问题，而不是应用代码产生的问题。如果已经将 2 个基本的任务和节拍中断起来，那么接下来添加应用任务是非常简单的。

可以使用各种不同的技术测试自己的移植工作，取决于用户个人在嵌入式系统方面的经验和对处理器的理解。这里通过 4 个步骤测试移植代码：

- 确保 C 编译器、汇编编译器及链接器正常工作；
- 验证 OSTaskStkInit()和 OSStartHighRdy()函数；
- 验证 OSCtxSw()函数；
- 验证 OSIntCtxSw()和 OSTickISR()函数。

第四章　嵌入式 TCP/IP 协议栈

在前面两章中已经分别介绍了嵌入式μC/OS-II 操作系统的使用、移植以及 TCP/IP 协议、局域网协议的检验内容。在本章中将介绍怎样使用 uC/OS-II 操作系统来编写小型的 TCP/IP 协议栈，使其可以接入到 Internet 互联网中，实现 TCP、UDP 的连接。

第一节　本系统的硬件结构

在本系统中，以 51 系列微控器作为主 CPU(RAM 和 ROM 存储值均为可寻址范围的最大值)，网络接口芯片使用的是 RTL8019AS。单片机寻址使用的是外部数据存储器的扩展方法。使用一 74573 来复用地址总线与数据总线。连接方法如图 4.1 所示。

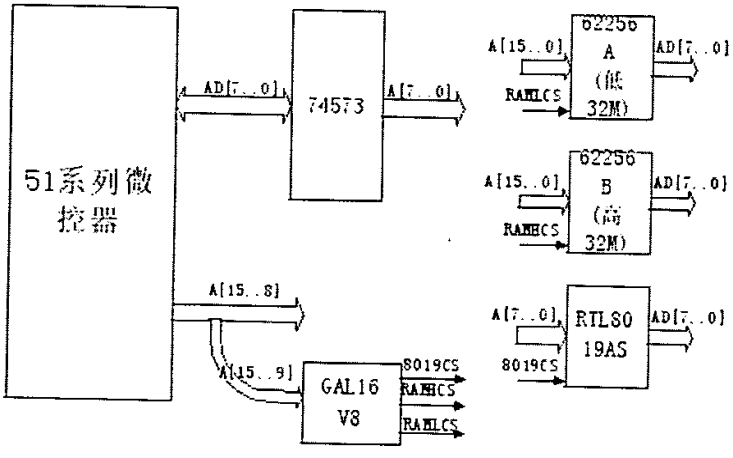

图 4.1　系统硬件连接图

AD[7..0]作为数据线(与地址线复用)连接两片 RAM 以及网络接口芯片，传送数据。16 位的地址线全部连接在两片 RAM 上，所以 RAM 的寻址空间到达了 64K(并没有全用)。在图中可以看到A[15..9]连接在GAL器件上，8019CS，RAMHCS，RAMLCS 分别作为网卡、高低地址内存芯片的使能端(低电平有效)控制芯片的使能。它们的逻辑关系如下：

8019CS=!(A15&A14&A13&A12&A11&A10&A9)

RAMLCS=A15

RAMHCS=!A15#A15&A14&A13&A12&A11&A10&A9

RAM 最高 512 字节不用，而是分配给 8019 控制器 64 字节(FE00-FE3F 或 FF00-FF3F)，微控器通过寻址这 64 字节的地址就可以完成对网卡芯片的操作。当 A15 为零时，低 32K 的内存空间被选中，系统可以对内存进行读写。同理对高 32K 内存空间读写。这样就完成了硬件系统的设计。

第二节　软件系统概述

在系统中一共建立了三个任务，在这里，把这三个任务分别称为网卡接收，TCP-Server 和 UDP-Server。三者的关系可以通过图 4.2 来表示

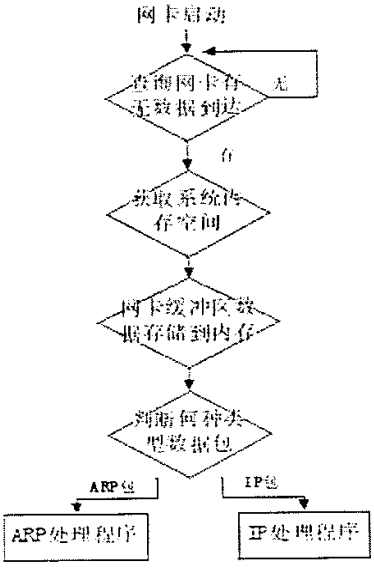

图 4.2　任务关系图

在三个任务中，TCP-Server 的优先级最高，UDP-Server 次之，网卡接收最低。当系统开始运行后，先将 TCP-Server 和 UDP-Server 初始化(具体如何进行将在后文介绍)，然后整个系统进入到运行状态。这时候由于没有数据到来，TCP-Server 和 UDP-Server 任务也就没有数据进行处理，这两个任务被内核挂起。只有网卡接收任务在一直运行(虽然它的优先级最低)。当有数据报到达的时候，

网卡接收程序首先判断为 ARP、IP 哪种数据包(本程序只考虑这两种类型的数据包，其他类型一律抛弃)，然后分别进行 ARP 和 IP 的处理，当判断出数据是 TCP 或 UDP 的数据包时，通过邮箱将数据所存在内存中的地址发送给两个任务，然后发送 TCP 或 UDP 任务的信号量，启动两个任务中的一个。任务被激活后，由于优先级比网卡任务高，所以马上获得 CPU 的使用权，进入运行状态，处理数据。

第三节 系统内存分配与管理

在整个系统中，内存的分配与管理是一个非常重要的部分。它决定着系统运行的效率的高低与速度的快慢。在本系统中，为微控器提供了 64KRAM 的数据存储空间，对于小型的 TCP 协议栈来讲，是可以满足使用要求的。对于内存的管理是按照 uC/OS-II 中的内存管理来进行的。[28]

在系统初始化时，调用函数 OSMemGet()创建内存分区，因为以太网的数据包最大的长度是 1580 字节，所以把每个分区的存储空间定为 1580 字节；同时为了防止数据包的溢出，可以在系统容量允许的条件尽量多分配内存数据块(本系统分配了 12 个)。这样就在数据存储区形成了一个内存链表。当有网络数据到达后，系统使用 OSMemGet()函数从内存链表中获得存储空间来存放数据。同时在数据处理完以后或发送出去以后，应该调用 OSMemPut()函数释放内存空间，否则就会造成内存溢出或系统崩溃。在系统中各任务之间对数据的处理就可以通过邮箱来发送存储数据的内存的地址，实现对数据的共享处理。

第四节 RTL8019AS 的控制

4.4.1 网卡总体介绍以及上电复位

RTL8019AS 的基本情况:输入输出地址：共 32 个，

地址偏移量为 00H--1FH。其中 00H—0FH 共 16 个地址，为寄存器地址；

10H—17H 共 8 个地址，为 DMA 地址；

18H—1FH 共 8 个地址，为复位端口；

对于 8 位的操作方式，上面的地址中只有 18 个是有用的：00H—0FH 共 16

个寄存器地址；

10H DMA 地址(10H－17H 的 8 个地址是一样的，都可以用来做 DMA 端口，只要用其中的一个就可以了)。

1FH 复位地址。(18H 到 1FH 共 8 个地址都是复位地址，每个地址的功能都是一样的，只要其中的一个就可以了，但实际上只有 18H，1AH，1CH，1EH 这几个复位端口是有效的，其他不要使用)跟复位有关的管脚是 RSTDRV，连接到 ISA 总线的 RSTDRV 的引脚上。RSTDRV 同时也是 ISA 总线的复位信号。RSTDRV 为高电平有效，至少需要 800ns 的宽度。给该引脚施加一个 1us 以上的高电平就可以复位。施加一个高电平之后，然后施加一个低电平。

寄存器：00H－0FH 共 16 个地址是寄存器地址。寄存器分成 4 页 PAGE0—PAGE3。

对网卡进行复位：这是网卡驱动程序的需要做的第一个内容，将网卡设置为跳线模式，而不是即插即用的模式。

网卡的复位子程序：

```
void rtl8019as_rst() //RTL8019AS 复位
   {
    unsigned char temp;
    delay_ms(400);//延时
    temp=reg1f;    //读网卡的复位端口
    reg1f=temp;    //写网卡的复位端口
    delay_ms(400);//延时
    }
```

上面的程序是网卡的热复位方式，还可以进行冷复位。所谓的冷复位就是使使网卡的 RSTDRV 引脚变成高电平，网卡是高电平复位的；然后再使网卡的 RSTDRV 引脚变成低电平，网卡上电复位完毕。

4.4.2　网卡初始化

```
void page(unsigned char pagenumber)
   {
   unsigned char data temp;
```

```
temp=reg00;
temp=temp&0x3B;
pagenumber=pagenumber;
temp=temp;
reg00=temp;
}
```

上文已经说过，在网卡中的寄存器是分页设置的，此程序即是用来转换寄存器的页面。reg00 命令寄存器：CR，command register，地址偏移量 00H，为一个字节。

PS1 和 PS0 这两个位用来选择寄存器页，PS1 PS0=00 时选择寄存器页 0=01 时选择寄存器页 1=10 时选择寄存器页 2=11 时选择寄存器页 3。

上面的程序的参数为 pagenumber,用来指定第几页。

```
temp=reg00 ;//读入命令寄存器的值
temp=temp&0x3f;//将高 2 位，即 PS1,PS0 清 0
pagenumber=pagenumber<<6;//将低 2 位移至高端
temp=temp|pagenumber; //写入高 2 位
reg00=temp; //设置第几页
```

• RD2,RD1,RD0 这 3 个位代表要执行的功能。

=001 读网卡内存

=010 写网卡内存

=011 发送网卡数据包

=1** 完成或结束 DMA 的读写操作

• TXP 这个位写入 1 时发送数据包，发完自动清零

• STA，STP 这两个位用来启动命令或停止命令

=10 启动命令

=01 停止命令

网卡的初始化：

结合上文所说，它完成的功能主要是初始化网卡。如下：

```
rtl8019as_rst(); //复位 8019
reg00=0x21; //使芯片处于停止模式,这时进行寄存器设置 停止模式下,将
```

不会发送和接收数据包

```
delay_ms(20);
page(0); //使用 0x4c－0x7f 为网卡的接收缓冲区。
reg0a=0x00; //使用 0x40-0x4B 为网卡的发送缓冲区，共 12 页，刚好可以存储 2 个最大的以太网包
reg0b=0x00; //延时 20 毫秒,确保芯片进入停止模式
reg0c=0xE0; //monitor mode (no packet receive)
reg0d=0xE2; //loop back mode
reg01=0x4C; //Pstart 接收缓冲区范围
reg02=0x80; //Pstop
reg03=0x4C; //BNRY
reg04=0x40; //TPSR 发送缓冲区范围
reg07=0xFF; /*清除所有中断标志位*/
reg0f=0x00; //IMR disable all interrupt
reg0e=0xC8; //DCR byte dma 8 位 dma 方式
page(1);    //选择页 1 的寄存器
reg07=0x4D; //CURR
reg08=0x00; //MAR0
reg09=0x00; //MAR1
reg0a=0x00; //MAR2
reg0b=0x00; //MAR3
reg0c=0x00; //MAR4
reg0d=0x00; //MAR5
reg0e=0x00; //MAR6
reg0f=0x00; //MAR7
WriteRtl8019NodeID(); //将网卡地址写入到 mar 寄存器
page(0);
reg0c=0xcc;      //将网卡设置成正常的模式,跟外部网络连接
reg0d=0xe0;
reg00=0x22;      //这时让芯片开始工作
reg07=0xff;      //清除所有中断标志位
PSTART 接收缓冲区的起始页的地址
```

PSTOP 接收缓冲区的结束页地址（该页不用于接收）

BNRY 指向最后一个已经读取的页（读指针）

CURR 当前的接收结束页地址（写指针）

网卡含有 16K 字节的 RAM，地址为 0x4000-0x7fff(指的是网卡上的存储地址，而不是 ISA 总线的地址，是网卡工作用的存储器)，每 256 个字节称为一页，共有 64 页。页的地址就是地址的高 8 位，页地址为 0x40—0x7f。这 16k 的 RAM 的一部分用来存放接收的数据包，一部分用来存储待发送的数据包。当然也可以给用户使用。

在程序中使用 0x40-0x4B 为网卡的发送缓冲区，共 12 页，刚好可以存储 2 个最大的以太网包。使用 0x4c－0x7f 为网卡的接收缓冲区，共 52 页。因此 PSTART=0x4c，PSTOP=0x80(0x80 为停止页，就是直到 0x7f，是接收缓冲区，不包括 0x80)刚开始，网卡没有接收到任何数据包，所以，BNRY 设置为指向第一个接收缓冲区的页 0x4c)

CURR 是网卡写内存的指针。它指向当前正在写的页的下一页。那么初始化它就应该指向 0x4c＋1＝0x4d。网卡写完接收缓冲区一页，就将这个页地址加 1，CURR＝CURR＋1。这是网卡自动加的。当加到最后的空页（这里是 0x80，PSTOP）时，将 CURR 置为接收缓冲区的第一页（这里是 0x4c，PSTART），也是网卡自动完成的。当 CURR＝BNRY 时，表示缓冲区全部被存满，数据没有被用户读走，这时网卡将停止往内存写数据，新收到的数据包将被丢弃不要，而不覆盖旧的数据。此时实际上出现了内存溢出。

而 BNRR 要由用户来操作。用户从网卡读走一页数据，要将 BNRY 加一，然后再写到 BNRY 寄存器。当 BNRY 加到最后的空页（0x80，PSTOP）时，同样要将 BNRY 变成第一个接收页（PSTART，0x4c）BNRY=0x4c。CURR 和 BNRY 主要用来控制缓冲区的存取过程，保证能顺次写入和读出。

当 CURR=BNRY+1（或当 BNRY＝0x7f,CURR=0x4c）时，网卡的接收缓冲区里没有数据，表示没有收到数据包。用户通过这个判断知道没有包可以读。当上述条件不成立时，表示接收到新的数据包。然后用户应该读取数据包，直到上述条件成立时，表示所以数据包已经读完，此时停止读取数据包。

TPSR 为发送页的起始页地址。初始化为指向第一个发送缓冲区的页，0x40。

RCR 接收配置寄存器，设置为使用接收缓冲区，仅接收自己的地址的数据包（以及广播地址数据包）和多点播送地址包，小于 64 字节的包丢弃（这是协议

的规定，设置成接收是用于网络分析)，校验错的数据包不接收。

TCR 发送配置寄存器，启用 crc 自动生成和自动校验，工作在正常模式。

DCR 数据配置寄存器，设置为使用 FIFO 缓存，普通模式，8 位数据传输模式，字节顺序为高位字节在前，低位字节在后(如果用 16 位的单片机，设置成 16 位的数据总线操作会更快，但 80c52 是 8 位总线的单片机)。

IMR 中断屏蔽寄存器，设置成 0x00，屏蔽所有的中断。设置成 0xff 将允许中断。

MAR0—MAR8 是设置多点播送的参数，由于不使用多点播送，所以只要保证网卡能正常工作就可以了,不必考虑。

PAGE2 的寄存器是只读的，所以不可以设置，不用设置。

4.4.3　网卡接收数据包

网卡 RAM 的结构。网卡 RAM 是以 256 字节为一页，是按页存储的结构，16 位 RAM 的地址高 8 位又叫页码。一般把前面的 12 页用来存放发送的数据包，后面的 52 页用来存放接收的数据包。

哪些页被用来做接收缓冲区由两个寄存器决定:

Pstart(page start register)

Pstop(page stop register)

PSTART,PSTOP 是 16 位 RAM 地址的高 8 位,也就是页码。

在本程序中:

PSTART=0x4c

PSTOP =0x80

这时网卡将使用 0x4c00-0x7fff 的 RAM 来存储接收到的数据包。PSTOP 是 0x80 而不是 0x7f，PSTOP 的意思是，从该页开始的页不能做为接收缓冲区。而 PSTART 的意思是从这一页开始做为接收缓冲区。

因此 PSTART=0x4c，PSTOP=0x7f+1=0x80，PSTOP 被设置为最后一页的页码+1。而页 0x40-0x4b 共 12 页做为发送数据缓冲区，为什么用 12 页,是因为最大的一个数据包是 1514 字节+4 字节校验。一个最大的数据包需要 6 页=256*6=1536 字节。

12 页可以放两个最大的包。把前 6 页 0x40-0x45 称为发送缓冲 1，接下来

的 6 页 0x46-0x4B 称为发送缓冲 2。这两个发送缓冲的作用是:

用户可以将数据包放在发送缓冲 1，然后启动发送。发送的过程中。如果用户还有数据包要发，那么这时把要发的数据包放在发送缓冲 2(一边发送，一边把下一包放到缓冲区里)，等到发完发送缓冲 1 的数据包就可以马上启动发送缓冲 2 的数据包。这样可以不断地进行发送数据。

设置了接收缓冲区之后,那么接收到的第一个数据包放在哪里由 CURR 决定。控制接收缓冲区的有两个寄存器 CURR，BNRY，CURR 是网卡写缓冲区的指针，指向此时要写的页。BNRY 是读指针，指向用户已经读走的页。

要知道网卡正在写哪一个字节的数据，读取 CLDA1,CLDA0 的值就知道了。初始化 CURR，BNRY 这两个指针时，使 BNRY=CURR-1 页。

```
BNRY=CURR-1;
if (BNRY<0x4c)BNRY=0x7f;
```

这就是两者的初始化的关系式。初始化 CURR=0x4d，表示第一个收到的数据包将存放在 0x4d00 地址开始的缓冲区。初始化 CURR=0x4c，表示第一个收到的数据包将存放在 0x4c00 地址开始的缓冲区。CURR 只初始化一次，以后都不用再去修改它。

网卡收到一个数据包，假如是一个短数据包，只要一页就可以存储的，那么这时候 CURR 将会被加 1。CURR 将等于 0x4d+1=0x4e。假如需要两页来存储，那么 CURR 将等于 0x4d+2=0x4f。如果 CURR 等于结束页 PSTOP,也就是 CURR>0x7f 时,CURR 将被重置成等于 PSTART=0x4c。CURR 是网卡内部自己控制的，用户不需要干预。网卡存储一定是按页存储，不满一页，也使用一页，下一包将用下一页开始存储。

4.4.4 对网卡 RAM 的读写

要接收和发送数据包都必须读写网卡的内部的 16k 的 RAM，必须通过 DMA 进行读和写。网卡的内部 RAM 是一块双端口的 16k 字节的 RAM。所谓双端口就是说有两套总线连接到该 RAM，一套总线 A 是网卡控制器读/写网卡上的 RAM，另一套总线 B 是单片机读/写网卡上的 RAM。总线 A 又叫 Local DMA,总线 B 又叫 Remote DMA。

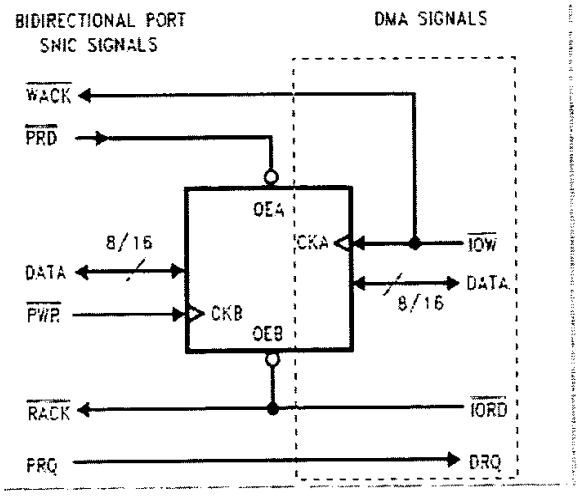

图 4.3　网卡 RAM 读写总线图

图 4.3 中虚线框住的部分为 Remote DMA，也就是单片机对网卡 RAM 进行读写的总线，对 8019 来说就是 ISA 总线。没有框住的部分(左边的部分)，就是 Local DMA，网卡控制器对网卡 RAM 进行读写的总线。

其中的地址总线没有画出来，只画了数据总线。实际在 RAM 的内部还有一些总线仲裁的逻辑，这里也没有画出来。所谓总线仲裁的逻辑就是为了实现两套总线都能进行对 RAM 的读写，而不互相冲突。网卡控制器读写网卡 RAM(Local DMA)的优先级比单片机读写网卡 RAM 的优先级要高。优先级要高的意思是：

- 当两者都要请求控制总线时. Local DMA 优先获得控制权；
- 高优先级的 Local DMA 可以中断 Remote DMA，而 Remote DMA 不能中断 Local DMA；
- 在 Remote DMA，也就是单片机对网卡 RAM 读写的过程进行中可以被 Local DMA 中断，Local DMA 中断 Remote DMA，然后进行 Local DMA 的数据传输，Local DMA 传输完毕之后继续刚才被中断的 Remote DMA，以完成 Remote DMA 的传输。

单片机的总线要比网卡的 DMA 总线慢很多。网卡的 DMA 总线大概在 10Mhz，而单片机的总线大概 1Mhz。所以在 Remote DMA 的过程中不需要特别的等待时序。DMA 有 8 位和 16 位两种。网卡支持这两种 DMA，在这里使用的是使用 8 位的 DMA。如下程序为读取 DMA 数据操作：

```
reg09=bnry; //读高地址页
reg08=4;    //读低地址页
```

```
reg0b=len>>8; //读数据高位
reg0a=len&0xff; //读数据低位
reg00=0x0a;    //read dma
for(ii=0;ii<len;ii++)
  {
    buf[ii]=reg10;
  }
  reg0b=0x00;
  reg0a=0x00;
  reg00=0x22; //dma complete page0
```

执行的结果是将网卡里的起始地址为 address 的共 count 个字节读入到 buffer[i]里。以下为写网卡 DMA 操作：

```
reg09=0x46;    //txdwrite highaddress
reg08=0x00;    //read page address low
reg0b=len>>8;  //read count high
reg0a=len&0xff; //read count low;
reg00=0x12;    //write dma, page0
for(ii=0;ii<len;ii++)
 {
   reg10=*(outbuf+ii);
 }
// 以下 3 句为中止 dma 的操作
reg0b=0x00; //read count high 中止 DMA 操作
reg0a=0x00; //read count low;
reg00=0x22; //complete dma page 0
```

通过以上两个程序就可以实现对 DMA 的操作，从而完成网卡数据的读写功能。

第五节 系统初始化

在系统启动之前，要进行初始化的过程。初始化的过程通过调用四个函数来完成

```
init_8019();
init_arp();
udpInit();
tcpInit()。
```

• init_8019()

结合上文所说，它完成的功能主要是初始化网卡。

• init_arp()：　它的功能主要是完成 ARP 缓存的初始化。

定义的结构体如下：

```
typedef struct
  {
  unsigned long ipaddr;
  unsigned char hwaddr[6];
  unsigned char timer;
  } ARP_CACHE;
```

在系统中共有 10 个缓存，设置一定的生存时间，实时进行更新。

• udpInit()：　它完成两项工作，一个是初始化 udps[i](udp 控制块)，另一个是建立 UDP 信号量和邮箱。

UDP 控制块的结构体定义如下：

```
typedef struct
{
unsigned long flags;//控制块标志位
unsigned long acceptFromAddr;//接受端地址
unsigned long theirAddr;//发送端地址
unsigned int theirPort;//发送端端口
unsigned int ourPort;//本地数据端口
unsigned int tos;//服务类型
UDP_HEADER xdata * udppayload;//定义数据地址指针
unsigned int len;//数据长度}
UDPCB;
```

在初始化时候，要将控制块的 flag 域均设为 0，表示没有被占用

```
for (i = 0; i < MAXUDP; i++)
```

```
{
  udps[i].flags = 0;
}
```

下面就要初始化信号量和邮箱，为任务的切换和数据传送作准备

```
if (!UdpSem)UdpSem = OSSemCreate(0);
if (!UdpMbox)UdpMbox = OSMboxCreate((void *)0);
• tcpInit();
if (!TcpSem)TcpSem = OSSemCreate(0);
if (!TcpMbox)TcpMbox = OSMboxCreate((void *)0);
```

在 TCP 连接过程中。没有设置 TCP 控制块(后面解释)，所以只需初始化信号量和邮箱。

第六节　网卡接收任务的结构与完成的功能

系统在上电初始化以后，在没有数据到来的情况下，TCP—Server 和 UDP—Server 任务都被挂起，网卡的任务在运行。网卡任务运行后，就调用 query_8019()函数查询网卡数据缓冲区。当发现有网络数据到达网卡以后，任务就调用系统函数 OSMemGet()从系统内存中取得一块内存存储区的使用权。然后将数据从网卡的数据缓冲区中 copy 到系统内存中进行处理。定义以太网报头的数据结构为

```
typedef struct
{
  unsigned char dest_hwaddr[6]; //本机物理地址
  unsigned char source_hwaddr[6]; //帧类型
  unsigned int frame_type; //对方主机物理地址
}
ETH_HEADER;
```

进行强制类型转换 eth = (ETH_HEADER xdata *)inbuf //inbuf 在这里指向存储数据的地址。下面判断 eth->frame_type(以太网数据类型字段)，只对 ARP 和 IP 两种类型的数据进行处理。

4.6.1　ARP 处理程序

ARP 处理程序义的 ARP 报头的数据结构体为

```
typedef struct
{
unsigned int hardware_type;
unsigned int protocol_type;
unsigned char hwaddr_len;
unsigned char ipaddr_len;
unsigned int message_type;
unsigned char source_hwaddr[6];
unsigned long source_ipaddr;
unsigned char dest_hwaddr[6];
unsigned long dest_ipaddr;
} ARP_HEADER;
```

首先要指向 ARP 的报头 arp =(ARP_HEADER xdata *)(inbuf + 14) // inbuf + 14 是指要剥离的 ETH 报头，判断 ARP 包头中的硬件类型和 IP 域值是否正确(是否为 ARP 的值)，然后在 ARP 高速缓存中查找是否有对应于发送者的 IP 地址。当没有查到的时候，判 IP 数据判断 ARP 报的目的 IP 地址是否为本机地址，同时更新 ARP 缓存。收到的 ARP 的数据包有两种类型

一种是 Request；

另一种是 Response Request 包是对方主机向本机发送 ARP 的请求信息，要求获得本机的物理地址。

这时本机就应发送包含本机物理地址的应答 ARP 包。Response 包发送的原理是这样：本机要发送 IP 数据包到对方主机，在 ARP 高速缓存中查找到目的主机的物理地址。如有，就封装在 IP 数据包头外的以太网包头上；如没有，则将此 IP 数据包处于等待状态，发送 ARP Request 报文。对方主机收到此报文后，会回送包含物理地址的 Response 报文。系统从中提取到对方主机的物理地址，然后发送出去处于等待状态的 IP 数据包，同时更新 ARP 缓存。

4.6.2　IP 数据处理的程序功能与结构

定义的 IP 数据报头的结构体：

```
typedef struct
{
unsigned char ver_len;
unsigned char type_of_service;
unsigned int total_length;
unsigned int identifier;
unsigned int fragment_info;
unsigned char time_to_live;
unsigned char protocol_id;
unsigned int header_cksum;
unsigned long source_ipaddr;
unsigned long dest_ipaddr;
}
IP_HEADER; //与处理 IP 数据相似，进行强制类型转换
ip=(IP_HEADER xdata *)(inbuf +14)/2
```

首先判断数据包的目的 IP 地址是否为本机的 IP 地址，然后计算 IP 数据包的包头长度。在 IP 包头里的“首部长度”域中的值表示的是单位数(最大表示 15 个单位)，一个单位表示 4 个字节。在 IP 分组的首部长度不是 4 字节的整数倍的时候，必须利用最后的字段加以填充(参阅前面介绍的 IP 包头的格式)。获得包头长度以后，就可以通过 IP 包头的“总长度”减去首部长度以获得数据块的长度。下一步就是计算校验和。在这里只检验数据包的首部，不包括数据部分。检验和的计算不采用 CRC 检验码。IP 检验和的计算方法是：将 IP 数据报首部看成 16bit 字的序列。先将校验和字段置零，将所有的 16bit 字相加后，将和的二进制反码写入检验和字段。收到数据报后，将首部的 16bit 字的序列再相加一次，若首部未发生任何变化，则和必为全 1。否则即认为出差错，并将次数据报丢弃。

具体的检验和实现算法如下：

```
for(i=0;i<(length)/2;i++)
{
 sum+=*ptr++; //ptr 为数据的首地址
 }
```

```
if (length&0x01) //表示长度为单数
 {
  sum=sum+((*ptr)&0xff00);
  }
  sum=(sum&0xffff)+((sum>>16)&0xffff); //高 16 位和低 16 位相加
if(sum&0xffff0000)
{  //表示有进位
  sum++;
}
```

当检验和是正确的时候，要判断 IP 数据报的版本是不是 IPV4 版本，本系统不接收 IPV6 的版本，如是则抛弃。

最后判断 IP 数据包中封装的运输层的数据包是何种类型：ICMP,TCP or ARP，分别进行处理。

• ICMP 程序处理的结构与功能

前面章节已经分析过 ICMP 协议的功能。它提供的一个很重要的报文就是 ICMP ECHO，也就是 PING 服务。在本系统中，也是只提供 PING 服务。ICMP 整个程序的服务流程。在 icmp_rcve()函数中，首先要进行校验和的检验，这里是检验整个数据部分(不包括包头)。然后判断发送的类型，只有在类型字段为 8 时(ECHO 请求)才回送数据，调用 ping_send()函数。在 ping_send()函数中，首先调用 OSMemGet() 获得一个内存分区，依据协议规定，设定 ICMP 包头类域的值：

```
ping_out->msg_type = 0;
ping_out->msg_code = 0;
ping_out->checksum = 0;
ping_out->identifier=ping_in->identifier;
ping_out->sequence =ping_in->sequence;
```

最后计算整个包中的数据部分的校验和的值，填入到包头校验和值域中，调用 ip_send()发送。

• UDP 数据的处理

当 UDP 的数据包进行完检验和等操作之后，系统要获取一个空闲的 UDPCB，通过判断 udp_flag 等标志可获得空闲信息。如果没有空闲的则退出，系统允许

最多建立 4 个 UDP 连接。获得空闲控制块以后，将所收到的 UDP 包的一系列特征与地址存储在 UDPCB 中

```
udps[ud].udppayload=(UDP_HEADER xdata *)inbuf;//数据的存储地址
udps[ud].acceptFromAddr = ip->source_ipaddr;     //本机地址
udps[ud].theirAddr = ip->source_ipaddr;
udps[ud].theirPort = udpheadpointer ->source_port;//本机端口
udps[ud].len = len;                              //数据长度
```

完成对 UDPCB 的操作以后，通过邮箱发送 UDPCB 的地址，同时发送信息量 UdpSem。这样 UDP-Server 的任务就可获得信号量从挂起态进入到运行态，同时可 以获得邮箱传送的地址，对数据进行操作处理。当处理结束后要释放 UDPCB。

• TCP 数据的处理 TCP 的程序处理流程与 UDP 相似，只是由于 TCP 的连接建立更复杂，所以没有设置 TCPCB 的控制块，而是直接通过邮箱发送 TCP 数据包存储的内存地址。同时发送信号量 TcpSem 启动 TCP-Server 任务，把对 TCP 的控制与处理全部交给 TCP-Server 任务来处理。

4.6.3　IP 数据发送的功能流程

IP 数据的发送，不只在网卡接收任务中存在，在 TCP-Server 和 UDP-Server 任务中同样存在，但是因为对于在 IP 层发送的数据来说使用的是同样的函数发送，所以在这里集中介绍。IP 数据包的发送是通过函数 ip_send()来完成的。要发送的数据从上层向下传递，交付给 ip_send()函数。在 ip_send()函数中的主要功能就是把数据封装上 IP 包头，然后再向下层传递。在函数中实现的功能主要是

```
ip = (IP_HEADER xdata *)(outbuf +14);//指向 IP 数据报首部地址
ip->ver_len = 0x45; //IPv4 有 20byte 长的首部
ip->type_of_service = 0; //服务类型
ip->total_length = 20 + len; //整个数据包的长度
ip->identifier = ip_ident++;
ip->fragment_info = 0; // 确认序列号
ip->time_to_live = 32; // 帧信息
ip->protocol_id = proto_id; // 最长生存时间
```

```
ip->header_cksum = 0; // 数据类型
ip->source_ipaddr = my_ipaddr; //检验和先置 0
ip->header_cksum=~cksum(outbuf+14,20);//填写本机 IP
ip->dest_ipaddr = ipaddr; //填写对方 IP
// 计算检验和，填入到 IP 包头的域中，IP 包头的长度设定为 20 字节(本
//系统附加信息不考虑，不会填在包头中)
etherLock();
hwaddr = arp_resolve(ip->dest_ipaddr);
//没有在 ARP 缓存中查找到物理地址，就要将 IP 数据处于等待状态
etherRelease();//对方的物理地址后在发送
if (hwaddr == NULL)
  {
  //wait 为专门定义的数据结构，用于存储处于等待状态的 IP 数据包的一
  //系列信息
    wait.buf = outbuf;
    wait.ipaddr = ip->dest_ipaddr;
    wait.proto_id = proto_id;
    wait.len = len;
    wait.timer = ARP_TIMEOUT;
    return;
  }
eth_send(outbuf, hwaddr, IP_PACKET, 20 + len);//向下层传递
```

eth_send()是封装以太网包头。只需把本机与对方主机的物理地址封装进去 就可以了。同时调用 send_frame()。此函数是微控器通过总线对 RTL8019AS 芯片 进行操作的函数，将数据通过网络发送出去。

第七节　UDP-Server 服务程序

在本系统中，UDP—Server 任务实现的功能就是回送数据，也就是对方主机给本机发送了什么数据，本机将不作任何修改，直接回送这些数据。通过此种功能就可以验证 UDP 的连接了。在 udpInput()函数中，发送邮箱值和信号量(前

已介绍)使 UDP—Server 任务获得信号量进入运行态。在函数 udpWrite()中，首先调用 OSMboxAccept()获得邮箱中的 UDPCB 的地址 值，然后通过地址获取发送来的数据包的数据内容，便于回送。这时候要调用要调用 OSMemGet()获取内存，为发送 UDP 数据开辟存储空间。在获取的这个内存块中，填写 UDP 数据报包头信息(由 UDPCB 获得)

```
udpHdr->source_port = udps[ud].ourPort; //端口号
udpHdr->dest_port = udps[ud].theirPort;
udpHdr->length = 8 + udps[ud].len;//长度
udpHdr->checksum = 0; // 将收到的数据报的数据拷贝过来
memcpy(&udpHdr->msg_data, (udps[ud].udppayload+ 42), udps[ud].len);
ip->dest_ipaddr = udps[ud].theirAddr;//IP 地址
ip->source_ipaddr = my_ipaddr;
// 计算检验和
sum = (ULONG)cksum(outbuf + 26, 8 + udps[ud].len);
//检验和增加伪首部
sum += (ULONG)0x0011;
sum += (ULONG)udps[ud].len;
// 如果有进位，处理
result = (unsigned int)(sum + (sum >> 16));
udpHdr->checksum = ~result;
ip_send(outbuf, udpsender_ipaddr, UDP_TYPE, udps[ud].len);
```

这时，对 UDP 数据报的操作已经完成，系统将数据交由下一层 IP 层处理。

第八节 TCP-Server 服务程序

TCP 是传输层协议中提供的一种可靠的比特流的传输机制。TCP 协议比起上面介绍过的 UDP 协议要复杂的多。有数据发送到本机的时候，首先在 IP 层进行处理。当对 IP 数据报的处理正确以后，如果是 TCP 数据，就交由 tcp_input()处理。在 tcp_input()中，主要完成的任务是检验 TCP 包头的数据、IP 地址等是否正确，然后发送信号量，使 TCP 任务进入运行态。[29]

4.8.1 tcp_rcve()函数功能与结构

此函数是 TCP 任务中的核心，它完成的主要功能是建立 TCP 状态机，完成 TCP 机制中的编号确认、运输连接管理以及状态机的建立与运行等工作，确保 TCP 任务的正常运行。本系统最多可以建立五个 TCP 的连接，当有一个新的数据包到达的时候，通过套接字来判断这个连接是否已经建立，进入到状态机的不同状态。

在 TCP 的传输机制中，它不是按传送的报文来编号。TCP 将所要传送的整个报文看成是一个个字节组成的数据流，然后对每一个字节编一个序号。在连接建立时，双方要商定初始序号。TCP 就将每一次所传送的报文段中的第一个数据字节的序号，放在 TCP 首部的序号字段中。TCP 的确认是对接收到的数据的最高序号表示确认。但返回的确认序号是已收到的数据的最高序号加 1。也就是说，确认序号表示期望下次收到的第一个数据字节的序号。同时，在 TCP 数据报的报头的 6 个比特的控制字段也是十分重要的，也要作判断。下面六个比特是说明报文段的控制比特：

(1)紧急比特 URG 当其为 1 时，表明此报文段应尽快传送，而不按原来的排队顺序来传送；

(2)确认比特 ACK 只有当其为 1 时，上面所说的确认号字段才有意义。为零时无意义；

(3)急迫比特 PSH 当其为 1 时，表明请求远地 TCP 将本报文段立即传送给其应用层，而不是等到整个缓冲区都填满了后再向上交付；

(4)重建比特 RST 当其为 1 时，表明出现严重的差错，必须释放连接，然后再重建运输连接；

(5)同步比特 SYN 它是在连接建立的时候使用。当 SYN=1 而 ACK=0 时，表明这是一个连接请求报文段。对方若同意建立连接，则应在发回的报文段中使 SYN=1 和 ACK=1。因此，同步比特 SYN 置为 1，就表示这是一个连接请求或连接接受报文，而 ACK 比特的值用来区分是哪一种报文；

(6)终止比特 FIN 用来释放一个连接；

当 FIN=1 时，表明欲发送的字节串已经发完，释放运输连接。TCP 的传输数据的连接建立是很复杂的，要采用一种叫“三次握手”的构成才能够实现。在主机 B 中运行服务器程序(与本系统相同)，在主机 A 中运行的是客户端程序。

主机 B 的服务器进程处于“听”的状态，不断检测是否有客户进程要发起连接请求。如有，即作出响应。主机 A 的 TCP 向主机 B 的 TCP 发出连接请求报文段，其首部中的同步比特 SYN 应置为 1，同时选择一个序号 x，表明在后面传送数据时的第一个数据字节的序号是 x。主机 B 的 TCP 收到连接请求报文段后，如同意，则发回确认。

在确认报文段中应将 SYN 置为 1，确认序号应为 x+1，同时也为自己选择了一个序号 y。主机 A 的 TCP 收到此报文段后，还要向 B 给出确认，其确认序号为 y+1。运行客户进程的主机 A 的 TCP 通知上层应用进程，连接已经建立。当运行服务器进程的主机 B 的 TCP 收到主机 A 的确认后，也通知其上层应用进程，连接已经建立。这就是“三次握手”的过程。同样，在数据传输结算后，通信的双方都可以发出释放连接的请求。它的释放的过程与“三次握手”在本质上是一致的，这里不再细述。

在 TCP 开始建立的时候“三次握手”以及释放连接的时候，TCP 的连接可能处于不同的状态。这样确定 TCP 连接可能处于的状态以及各种状态可能发生的变迁。每一个方框就是 TCP 可能具有的状态。状态之间的箭头表示可能发生的状态变迁。箭头旁边的字表明是什么原因引起的这种变迁，或表明发生状态变迁后又出现什么动作。下面对此作简单的解释：我们从服务器进程来分析状态图的变迁(这也是本系统所实现的功能)。服务器进程发出被动打开，进入听状态 LISTEN。当收到 SYN 置为 1 的连接报告后，发送确认 ACK，并且报文中的 SYN 也置为 1，然后进入 SYN_RCVD 状态。在收到三次握手中的最后一个确认 ACK 时，就转为 ESTABLISHED 状态，进入数据传送阶段。

当客户进程的数据已经传送完毕，就发送出 FIN 置为 1 的报文给服务器进程，进入 CLOSE_WAIT 状态。服务器进程发送 FIN 报文段给客户进程，状态 LAST_ACK 变为状态。当收到客户进程的 ACK 时，服务器进程就释放连接，删除连接记录，状态回到原来的 CLOSED 状态。这时就可以发送和接受数据了，TCP 连接建立和关闭的发生变化。在上面的介绍中提到在服务器的运行程序中要建立 TCB(传输控制模块)，在本系统中 TCB 的结构体的建立如下所示：

```
typedef struct {
  unsigned long ipaddr;//地址
  unsigned int port;//端口
  unsigned long his_sequence;//对方序列号
```

```
  unsigned long my_sequence;//本机序列号
  unsigned long old_sequence;//储存序列号
  unsigned long his_ack;//对方确认
  unsigned char state; //TCP 连接所处状态
  char query[20];
  unsigned int tos;//服务类型
  TCP_HEADER xdata* tcppayload;//数据存储地址
  unsigned int len;
} TCPCB
```

TCPCB 记录了每一个连接的地址、端口、序列号以及所处的状态。通过记录的状态来判断连接应该进入到状态机中的何种状态。下面仅以 STATE_LISTEN 状态的代码为例进行说明：

```
case STATE_LISTEN:
// 如果接受的数据只有 SYN 而没有 ACK 则说明是请求建立的第一次发送的
数//据包，进行处理
if ((tcp->flags & FLG_SYN) && ((tcp->flags & FLG_ACK) == 0))
{
//获得对方的序列号并且产生本机的序列号，同时在 TCPCB 记录
tcps[nr].ipaddr = ip->source_ipaddr;
// 建立此次连接的序列号，注意，关中断 OS_ENTER_CRITICAL();
tcps[nr].port = tcp->source_port; // 连接的信息
tcps[nr].state = STATE_LISTEN;     //状态为 LISTEN
tcps[nr].his_sequence = 1 + tcp->sequence;
//获得对方的序列号并且产生本机的序列号，同时在 TCPCB 记录
tcps[nr].his_ack = tcp->ack_number;
// 建立此次连接的序列号，注意，关中断 OS_ENTER_CRITICAL();
tcps[nr].my_sequence = initial_sequence_nr;
//因为本机的序列号总是指向下一个要被发送的字节的序号
initial_sequence_nr += 64000L;
OS_EXIT_CRITICAL();
// 而发送来的 ACK 已经占用了一个字节，所以在这里应该加 1
```

```
tcps[nr].my_sequence++;
//将状态机的状态改变为接受 SYN 确认
tcps[nr].state = STATE_SYN_RCVD;
//发送方发送的并不是要求建立连接的数据，抛弃，并通知对方
}
else
{
tcps[nr].ipaddr =0;
tcp_send(FLG_RST, 20, NO_CONNECTION); //重新建立连接
}
break; //退出状态机
```

这只是一个 LISTEN 状态的变换，其它状态的变换是与它相似的。当状态机处于 ESTABLISHED 状态的时候，这时可以传送数据为上层提供服务。在本系统中是通过调用 tcp_process()函数来实现的。

4.8.2　tcp_process()函数的功能和结构

tcp_rcve()是为上层提供服务是由 tcp_process()函数来完成的，在这里设计的上层完成的功能是这样的：

对方向本机发送一个数据包，经过 ip_input(),tcp_rcve()等函数后交由 tcp_process()函数来处理。在 tcp_process()函数中，首先取出数据包中的数据，检验其是否为字符串“hello”。当为是的时候，给对方主机发送字符串“This is right data”；当不是的时候发送“This is wrong data”。

在程序中，使用 strcmp()函数来判断接收到的数据是否为字符串“hello”。然后调 OSMemGet()获得内存分区，发送数据报。在这里做的工作主要是填写 TCP 报头和 I 报头内容，下面给出代码进行简单的分析：

```
tcp->source_port = 1000;
tcp->dest_port = 1024;  //设置本机和对方主机的端口号
tcp->ack_number=tcps[nr].his_sequence; //填写 TCP 报头中的序列号
tcp->sequence =tcps // 填写报头控制 bit 内容 tcp->
flags = (headerlength << 10) | FLG_ACK;
```

```
tcp->window = 1024
tcp->checksum = 0; //窗口设置固定值
tcp->urgent_ptr = 0; // 目的主机的 IP 和本机的 IP 地址
ip->dest_ipaddr = tcps[nr].ipaddr;
ip->source_ipaddr = my_ipaddr; //计算总的校验和
sum =(unsigned long)cksum(outbuf + 26, 8 + 20+datalen);
//包括虚首部一部分的值
sum+=(unsigned long)0x0006; // 还应加上虚首部其他的值进行计算
sum+=(unsigned long)0x2a; //此包总的长度＝首部＋数据
result = (unsigned int)(sum + (sum >> 16)); // 填写 TCP 首部值域
tcp->checksum = ~result;
ip_send(outbuf, dest, TCP_TYPE, length);//调用 ip_send()发送
```

这样，就完成了数据的发送。

4.8.3　tcp_send()函数的功能与结构[30]

在4.8.1节，曾经提到过在状态机的任何一个状态(这里不包括ESTABLISHED状态)都会发送TCP的数据。在这里的发送，就是调用tcp_send()函数来完成，因为它只是发送联络信息，不如“三次握手”的数据等，并不发送实际的数据信息。tcp_send()函数数据的发送与以上讲述过的区别不大。主要通过填写TCP报头中控制bit发送信息来进行联络，确保TCP运输管理的正常运行。

第五章　总结与展望

第一节　论文的工作总结和创新点

5.1　主要工作总结

本论文兼顾了小容量和通用性的要求，它可以在多种硬件平台上实现。多次测试表明。用论文提供的设计方案设计出的协议软件代码量小结构清晰易于移植和扩充。可以使用μC/OS-II 提供的 API 和各种服务供软件或者硬件开发，还可以进一步在此平台上开发各种基于网络的应用。

在进行网络数据测试的过程中，对数据包的抓取是非常重要的。因为必须得到网络数据，才能对数据进行分析，进而确认发送或接收的数据是否正确。测试中，使用的是 IRIS 软件来进行网络数据的监控。它可以抓取局域网内的所有数据包，显示包头信息以及数据的内容。通过它就可以抓取本机发送的所有数据包以及所有发送给本机的数据包。然后可以通过对它们的分析来判断程序中存在的问题。同时，在本系统中，设置了串口。在调试的过程中，可以把调试信息通过串口向外进行输出。这样就可以跟踪程序的运行状态。

本系统完成的是 TCP 和 UDP 的连接。在进行 UDP 的测试的时候，对方主机向本机发送数据包，本机回送同样的数据；TCP 连接是对方主机发送一定的数据，比如字符串“Is this server?”，本机收到后作判断，如果是，回送一定的数据；不是，回送另外的数据，这些数据都上可以自行设定的。经过测试，这两种连接都可以顺利建立，实现预先设计的功能。

5.2　创新点介绍

基于嵌入式操作系统的嵌入式系统开发，优点是显而易见的，但也有一些不利于它的推广应用的方面，那就是对开发人员有更高的要求。往往需要开发人员具备很强的计算机理论知识，对操作系统非常熟悉，这在一定程度上阻碍了嵌入式技术的发展。为解决这一问题，本论文实现了基于 uC/OS-II 和 TCP/IP

的嵌入式网络控制系统的开发平台。将 uC/OS-II 成功的移植到嵌入式系统，使得开发人员可以不必考虑硬件细节和驱动程序，直接调用 μC/OS-II 提供的 API 和各种服务，这样就可以减少软件和硬件设计的工作量，开发人员可进行更广泛的进行应用研究而不必深入了解系统本身。

第二节 有待改进的方面

由于时间的关系，本系统只实现了 TCP/IPD 的一些基本的 TCP 和 UDP 连接功能。以后的同学还可以在此基础上继续开发其他应用，在系统上开发新的功能。具体可以在如下几个方面考虑进行：

5.2.1 发送端的程序

本系统目前只有服务器端的程序，必须要对方主机向本机提起请求，双方才能建立连接。以后可以进一步考虑在本系统中加入客户端的程序进程，这样从本机也可以向其他主机直接发起连接。我在论文进行过程中，曾经实验了多任务的客户端的程序，并且也实验成功。以后就可以在多任务进行的过程中把此进程加入。

5.2.2 与高层协议的通信

在本系统中只是建立 TCP 和 UDP 的连接，实现用两种方式进行数据传送，并没有考虑与应用层协议的连接问题(无论是 TCP 方式还是 UDP 方式)。现在，两种连接已经建立了起来，就可以考虑应用层的协议了。比如可以通过 TCP 连接建立简单邮件传送协议 SMTP，还可以进行远程登录 TELNET 等。实现应用层的协议，就可以考虑向工业控制或者机顶盒等其它设备的移植工作了。

5.2.3 引入图形界面

图形用户界面也是嵌入式系统人机界面的一个趋势，如何引入 GUI 引擎并解决基于任务级的调试方法，都可以作进一步的研究，便于用户更方便的使用本系统。

致谢

在论文就要脱稿的时候，我要衷心感谢我的导师张立民老师！本文是在张老师的精心指导和关心下完成的。三年的学习生活中，张老师在学术上给我耐心的指导，培养我独立思考的能力，引领我走入了科研领域，培养了我的科研兴趣，提高了我的科研能力；在为人处世方面，给了我积极地引导，培养我踏实认真的学习态度与执着的追求精神。

同时还要感谢实验室和院系的老师。三年来，院系的多位老师都在各方面给予了我很大的帮助和鼓励，在此，向杨文霞老师、李维祥老师、阎少林老师、王繁珍老师、孙桂玲老师等信息技术科学学院的老师表示感谢！

论文在进行过程中得到了吕超同学的极大帮助，本课题是与她合作完成的，成稿时她帮我分析了很多材料和程序，使我避免了很多错误。还要感谢实验室中的同学，梁磊，鞠浩，张帆他们，他们给了我友谊与帮助，帮助我分析实验中遇到的各种问题，给出好的建议，能在一个友爱互助的学习氛围中不断吸取同学长处，给了我很多好的学习参考。

最后谢谢妻子王胜华给我的理解和支持，使我能专心致志，努力学习。

参考文献

[1] 谢希仁著，《计算机网络(第二版)》北京：电子工业出版社 1999.4

[2] Rene Trenado 著，《8 位微控制器可用于实现因特网协议》 北京：人民邮电出版社，2001-07-29

[3] 蔡德聪著，《工业控制计算机实时操作系统》北京：清华大学出版社，1999

[4] 《TCP/IP 协议族（第 2 版）》 Behrouz A.Forouzan,Sophia Chung Fegan 著 北京：清华大学出版社

[5] 《TCP/IP 网络与协议 》作者：兰少华 杨余旺 吕建勇 北京：清华大学出版社

[6] 《TCP/IP 协议及网络编程技术》作者：罗军舟 黎波涛 杨明 吴俊 黄健 北京：清华大学出版社

[7] 赵刚著，《TCP/IP 宝典》 第一版 2002，北京：电子工业出版社

[8] 张文著，《TCP/IP 基础》 第七版，2003，北京：电子工业出版社

[9] 李津生 洪佩琳著， 《下一代 Internet 的网络技术》北京：人民邮电出版社 2001.3

[10] W.Richard Stevens 著，范建华等译 《TCP/IP 详解，卷 1：协议》北京：机械工业出版社

[11] Tim Parker 著 前导工作室译 《TCP/IP 技术大全》北京：机械工业出版社 2000.7

[12] uC/IP - TCP/IP for microcontrollers http://ucip. sourceforge. net/

[13] lwIP-A Lightweight TCP/IP Stack http://www.sics.se/~adam/lwip/2

[14] 汤子流，哲凤屏，汤小丹著，《计算机操作系统》西安：西安电子科技大学出版社，1999

[15] 探矽工作室，《嵌入式系统开发圣经》北京：中国青年出版社，2002. 1.

[16] 林永仁著，《嵌入式系统项目分析入门与实践》北京：中国铁道出版社，2004

[17] 许海燕，付炎著，《嵌入式系统技术与应用》，北京：北京机械工业出版社，2002

[18]《嵌入式系统及单片机应用》 作者：李诚人等 北京：清华大学出版社

[19] Bart Broekman, Edwin Notenboo 著，张君施等译，《嵌入式软件测试》北京：电子工业出版社，2004

[20] 孙涵芳 徐爱卿著，《MCS-51/96 系列单片机原理及应用（修订版）》北京：北京航空航天大学出版社 2001.3

[21] 陈爱第著，《Protel 99 实用培训教程》北京：人民邮电出版社 2001.7

[22] 谭浩强著，《C 程序设计(第二版)》北京：清华大学出版社 1999.12

[23] 肖踞雄著，《单片机与嵌入式系统应用》北京：人民邮电出版社 2003

[24] [美]Jean J. Labrosse 著，邵贝贝等译 《嵌入式实时操作系统 μ C/OS-Ⅱ (第二版)》北京：北京航空航天大学出版社 2003.5

[25] 何轩，夏应清，李祥著，《基于以太网的的嵌入式系统网络接口卡的设计》北京：电子工业出版社

[26] 陈明计 周立功等著，《嵌入式实时操作系统 Small RTOS51 原理及应用》，北京：北京

航空航天大学出版社

[27] 胡大可著，《基于单片机 8051 的嵌入式开发指南》作者：胡大可 北京：电子工业出版社

[28]《嵌入式系统原理与开发》作者：夏靖波 王航 陈雅蓉 西安：西安电子科技大学出版社

[29] REALTEK 公司. RTL8019AS Datasheet

[30] 侯伯亨 顾新著，《VHDL 硬件描述语言与数字逻辑电路设计—电子工程师必备知识》西安：西安电子科技大学出版社 1999.1

个人简历

康小军，男，中共党员，陕西人,生于1976年4月；
1993年入伍入学就学于原解放军通信工程学院，学习无线通信工程专业，获学士学位；
1997年本科毕业分配服役于解放军68321部队，从事通信保障工作；
2003至2006年就读于南开大学信息技术科学学院，学习通信与信息系统专业；
在部队期间参加过野战地域通信网的设计施工；在研期间参加过智能氧分压仪的设计。

TCP/IP协议在嵌入式系统上的实现

作者： 康小军
学位授予单位： 南开大学

参考文献(30条)

1. 谢希仁.陈鸣.张兴元 计算机网络 1999

2. Rene Trenado 8位微控制器可用于实现因特网协议 2001

3. 蔡德聪 工业控制计算机实时操作系统 1999

4. BehrouzA Forouzan.Sophia Chung Fegan TCP/IP协议族

5. 兰少华.杨余旺.吕建勇 TCP/IP网络与协议

6. 罗军舟.黎波涛.杨明.吴俊.黄健 TCP/IP协议及网络编程技术

7. 赵刚 TCP/IP宝典 2002

8. 张文 TCP/IP基础 2003

9. 李津生.洪佩琳 下一代Intemet的网络技术 2001

10. W Richard Stevens.范建华.胥光辉.张涛.等.谢希仁 TCP/IP详解,卷1:协议

11. Tim Parker.前导工作室 TCP/IP技术大全 2000

12. uC/IP-TCP/IP for microcontrollers

13. IwIP-A Lightweight TCP/IP Stack

14. 汤子流.哲凤屏.汤小丹 计算机操作系统 1999

15. 探矽工作室 嵌入式系统开发圣经 2002

16. 林永仁 嵌入式系统项目分析入门与实践 2004

17. 许海燕.付炎 嵌入式系统技术与应用 2002

18. 李诚人 嵌入式系统及单片机应用

19. Bart Broekman.Edwin Notenboo.张君施.张思宇.周承平 嵌入式软件测试 2004

20. 孙涵芳.徐爱卿 MCS-51/96系列单片机原理及应用 2001

21. 陈爱第 Protel 99实用培训教程 2001

22. 谭浩强 C程序设计 1999

23. 肖踞雄 单片机与嵌入式系统应用 2003

24. Jean J Labrosse.邵贝贝 嵌入式实时操作系统μC/OS-II 2003

25. 何轩.夏应清.李祥 基于以太网的的嵌入式系统网络接口卡的设计

26. 陈明计.周立功 嵌入式实时操作系统Small RTOS51原理及应用

27. 胡大可.李培弘.方路平 基于单片机8051的嵌入式开发指南

28. 夏靖波.王航.陈雅蓉 嵌入式系统原理与开发

29. REALTEK公司 RTL8019AS Datasheet

30. 侯伯亨.顾新 VHDL硬件描述语言与数字逻辑电路设计—电子工程师必备知识 1999

本文读者也读过(10条)

1. 龚海刚 嵌入式信息终端TCP/IP协议栈的实现[学位论文]2003

2. 张大鹏 一种基于μC/OS-II的TCP/IP协议栈的实现[期刊论文]-科技信息2009(34)

3. 程晶晶 基于以太网技术的现场总线装置研究[学位论文]2002
4. 杜广微.张军国.韩宁.DU Guang-wei.ZHANG Jun-guo.HAN Ning 基于TCP/IP协议栈的嵌入式以太网终端设计[期刊论文]-航空计算技术2007,37(3)
5. 王新建 微操作系统中轻量级TCP/IP协议的研究与实现[学位论文]2005
6. 夏卓群 基于QoS的网络拥塞控制研究[学位论文]2006
7. 高小国 TCP/IP协议在传统终端上的实现[学位论文]2006
8. 陈昶 嵌入式MP3流IP数字广播终端的研究与设计[学位论文]2007
9. 黄志平.段国云 基于μC/OS-Ⅱ的TCP/IP协议的嵌入式驱动程序设计[期刊论文]-福建电脑2009(12)
10. 张东 有线/无线混合网络下的TCP研究[学位论文]2003

本文链接：http://d.g.wanfangdata.com.cn/Thesis_Y968700.aspx