基于嵌入式系统的电子邮件及聊天功能的设计和实现

摘 要

随着 IT 技术的日益发展，嵌入式设备的功能越来越强大。 嵌入式设备决不再只是充当一个小小的个人信息管理工具了，更重要的是可以上网浏览、收发邮件，在线聊天等。然而就目前国内市场来说，并不是所有的嵌入式设备都具备以上所有的功能，即使具备，也可能由于缺乏相应的服务而不能实现。但可以预见，嵌入式设备发展的趋势和潮流就是计算、通信、网络、存储等功能的融合。

目前市场上大多数的嵌入式系统都是商业化产品，价格昂贵，因此在开发过程中有必要使用一个模拟器，使得在 PC 上可以模拟一个嵌入式设备。在 PC 机上可以方便的开发和测试程序，并且可以帮助工程师快速的进行分析和改进，提高了工作效率和降低了项目的开发成本。

在嵌入式操作系统的领域中，Linux 以其特有的魅力得到了众多开发商的亲睐。Linux 这些特性包括可以移植到多个同结构的 CPU 和硬件平台上，有很好的稳定性以及各种性能的升级能力。

本文一开始对嵌入式技术作了简洁的介绍，并概括阐述了当前嵌入式系统的开发环境的类型以及各自的特点。然后针对大家都很熟悉的 TCP/IP 协议进行了简单的描述。在第二章，针对介绍本论文所应用到嵌入式系统开发环境作了一个简单的介绍。第三章主要介绍了在一个嵌入式系统模拟器下设计和实现电子邮件功能。并且实现的系统具有高实时性和高效率的特点，满足嵌入式系统的特点。第四章是对在上述模拟器上设计和实现聊天功能的描述。第五章主要介绍了在嵌入式 Linux 下设计和实现电子邮件功能。在最后的第六章是对本论文的总结和进一步工作的展望。

关键词 嵌入式系统 模拟器 TCP/IP 电子邮件

The Design and Implementation of Email and Chat Function Based on Embedded System

Abstract

With the development of the Information Technology, the function of the embedded device is becoming more and more powerful. It is not only used as personal information manager, but also used to browse the websites, receive and send email, chat and so on. And in the home market at the present time, not all the devices have these functions. But it can be foreseen that it is a tendency for the embedded device to be equipped the abilities of computer, communication, network, and storage.

At present, most of the embedded systems are very expensive. In order to cut down the cost of development, it is very necessary to utilize a simulator to simulate the embedded system's actions at first. Also with the simulator, the developer can develop, test and analyze the code quickly.

In the embedded OS region, Linux is welcome by many producers because Linux has following characters such as easily portability, supporting many different CPU architectures, stability and easily being upgraded and open resource policy.

At the beginning of the thesis, embedded technology is introduced compactly, especially emphasis on the types of embedded system develop environments. Then the well-known TCP/IP protocol is simple discussed. In the second chapter, the develop environments used in the thesis are stated. In the third chapter, the design and implement of email in one embedded system's simulator are described. These modules that designed are so efficient and real-time that it is suit for an embedded system. In the forth chapter, the design and implement of email in the embedded system's simulator are described. In the fifth chapter, the design and implement of email function in the embedded Linux are described. At the end, a summary of this thesis is made and some future expectations are presented in the sixth chapter.

Key words Embedded System, Simulator, TCP/IP, Email

声　明

本人声明所呈交的学位论文是在导师的指导下完成的。论文中取得的研究成果除加以标注和致谢的地方外，不包含其他人已经发表或撰写过的研究成果，也不包括本人为获得其他学位而使用过的材料。与我一同工作过的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

本人签名：

日　　期：2004.2.9

第一章 引 言

随着 IT（信息）技术的飞速发展，特别是互联网的迅速普及，3C（计算机、通讯、消费电子）合一的加速，数字化时代已经来临。嵌入式系统是数字化时代的一大主流产品，世界各国在此领域开始了激烈竞争，以争取获得主导地位。嵌入式系统是数字化产品的核心。

为适应嵌入式分布处理结构和应用上网需求，面向 21 世纪的嵌入式系统要求配备标准的一种或多种网络通信接口。针对外部联网要求，嵌入设备必需配有通信接口,相应需要 TCP/IP 协议族软件支持；由于家用电器相互关联(如防盗报警、灯光能源控制、影视设备和信息终端交换信息)及实验现场仪器的协调工作等要求，新一代嵌入式设备还需具备 IEEE1394、USB、Bluetooth 或 IrDA 通信接口，同时也需要提供相应的通信组网协议软件和物理层驱动软件。为了支持应用软件的特定编程模式，如 Web 或无线 Web 编程模式，还需要相应的浏览器，如 HTML、WML 等。可以想象，如果一个体积很小的嵌入式系统能够进行收发邮件，在线聊天，HTTP 通信，将是一件多么美妙的事情啊!

但是现在大多数的嵌入式系统都是商业化产品，价格昂贵。所以嵌入式系统开发前期投入比较大，开发工具投入高。为了改变这种状况，有必要设计一个模拟器，使得在 PC 上可以模拟一个嵌入式设备。在 PC 机上可以方便的开发和测试程序，并且可以帮助工程师快速的进行分析和改进，提高了工作效率和降低了项目的开发成本。

随着自由软件理念日益为大众广为接受，一批自由软件在 IT 的各个领域取得了巨大成功，Linux 是其中最具代表性的一个。其自由、开放的特性吸引了大批软件厂商和独立开发人员。多年的发展和业界同仁的共同努力已使其成为高效、稳定、低成本的操作系统。而且由于 Linux 开放源代码，任何人都可以对其进行裁减、修改，以适应自行开发的需要。

1.1 嵌入式系统概述

1.1.1 嵌入式系统的定义

嵌入式系统是指同时将操作系统和功能软件集成于计算机硬件系统之中的一种系统。简单的说就是应用软件、操作系统和硬件系统一体化，类似于 BIOS 的工作方式。具有代码小，高度自动化，响应速度快的特点。特别适合于要求实时的和多任务的体系。

嵌入式系统最初是指用以控制设备的计算机，通常是在设备内部，为了控制设备行为或是嵌入在其它系统中的一种专用软件和硬件。它一旦启动就执行某一特定的程序，中间无需人工干预，直到关机为止。但通常要求具有实时响应能力，一般不要求复杂的用户界面，甚至不要求支持键盘、显示器、串行口、硬盘等外设接口，也不需用户进行二次开发。它被广泛地用于仪器仪表、工业控制设备、电梯、程控交换机、微波设备、交通灯、家用电器等设备中。近年来，随着信息技术的飞速发展，嵌入式系统具有了新的内涵，同时萌生了许多形态各异的接入设备，如手持电脑、可上网的无线移动手机、机顶盒、家庭网关、可上网的电视机、可上网的车载盒、智能家用电器等等。相应地对嵌入式软件也提出了与最初不同的要求。

1.1.2 嵌入式系统的特点

嵌入式系统的所有特点都是体现在“嵌 入”这个词 上。既然要嵌入就必须要要嵌入的主体和嵌入的客体。从这些特点中也要能区分出嵌入式系统与普通通用的系统的差别。总体来说，嵌入式计算机系统同通用型计算机系统相比具有以下特点：[30]

(1) 嵌入式系统通常是面向特定应用的。嵌入式 CPU 与通用型的最大不同就是嵌入式 CPU 大多工作在为特定用户群设计的系统中，它通常都具有低功耗、体积小、集成度高等特点，能够把通用 CPU 中许多由板卡完成的任务集成在芯片内部，从而有利于嵌入式系统设计趋于小型化，移动能力大大增强，跟网络的耦合也越来越紧密。

(2) 嵌入式系统是将先进的计算机技术、半导体技术和电子技术与各个行业的具体应用相结合后的产物。这一点就决定了它必然是一个技术密集、资金密集、高度分散、不断创新的知识集成系统。

(3) 嵌入式系统的硬件和软件都必须高效率地设计，量体裁衣、去除冗余，力争在同样的硅片面积上实现更高的性能，这样才能在具体应用中对处理器的选择更具有竞争力。

(4) 嵌入式系统和具体应用有机地结合在一起，它的升级换代也是和具体产品同步进行，因此嵌入式系统产品一旦进入市场，具有较长的生命周期。

(5) 为了提高执行速度和系统可靠性，嵌入式系统中的软件一般都固化在存储器芯片或单片机本身中，而不是存贮于磁盘等载体中。

(6) 嵌入式系统本身不具备自行开发能力，即使设计完成以后用户通常也是不能对其中的程序功能进行修改的，必须有一套开发工具和环境才能进行开发。

1.1.3 嵌入式系统开发需要的开发工具和环境

通用计算机具有完善的人机接口界面，在上面增加一些开发应用程序和环境即可以对自身的开发。而嵌入式系统本身不具备自我开发能力，即使设计完成后，用户通常也是不能对其中的程序功能进行修改的，必须有一套开发工具和环境才能进行开发，这些工具和环境一般是基于通用计算机上的软硬件设备以及各种逻辑分析仪，混合信号示波器等。

嵌入式处理器是一个复杂的高科技系统，要在短时间内掌握并开发出所有功能是很不容易的，而市场竞争则需要产品能够快速上市，这一矛盾需要嵌入式处理器能够有容易掌握和使用的开发工具平台。提高用户和程序员的时间投入回报率。目前嵌入式系统的开发工具平台主要包括下面几类。

(1) 实时在线仿真系统 ICE(Ir- Circuit Emulator)

直到计算机辅助设计非常发达的今天，实时在线仿真系统仍是进行嵌入式系统调试最有效的开发工具。ICE 首先可以通过实际执行，对应用程序进行原理性检验，排除以人的思维难以发现的设计逻辑错误。ICE 的另一个主要功能是在应用系统中仿真硬件的实时执行，发现和排除由于硬件干扰等原因引起的异常执行行为。此外，高级的 ICE 带有完善的跟踪功能，可以将应用系统的实际状态变化，硬件的状态变化的反应等以一种录像的方式连续纪录下来，以供分析，在分析中优化控制过程。很多机电系统难以建立一个精确有效的数学模型，或建立模型需要大量人力，这时采用 ICE 的跟踪功能对系统进行纪录和分析是一个快而有效的方法。

(2) 高级语言编译器(Compiler)

C 语言作为一种通用的高级语言，大幅度提高了嵌入式系统工程师的工作效率，使之能够充分发挥出嵌入式处理器日益提高的性能，缩短产品进入市场的时间。另外 C 语言便于移植和修改，使产品的升级和继承更迅速，更重要的是采用 C 语言编写的程序易于在不同的开发者之间进行交流，从而促使嵌入式系统开发的产业化。

区别于一般计算机中的 C 语言编译器，嵌入式系统中的 C 语言编译器要专门进行优化，以提高编译效率。优秀的嵌入式系统 C 编译器代码长度和执行时间仅比汇编语言编写的同样功能程序长 5~20%。编译质量的不同，是区别嵌入式 C 编译器工具的重要指标。而 C 编译器与汇编语言工具相比残余 5~20%效率的差别，完全可以由现代微控制器的高速度、大存储空间以及产品提前进入市场的优势来弥补。

新型的微控制器指令的速度不断提高，存储器空间也相应加大，已经达到甚至超过目前的通用计算机中的微处理器，为嵌入式系统工程师采用过去一直不敢

问津的 C++语言创造了条件。C++语言强大的类，继承等功能更便于实现复杂的程序功能。但是 C++语言为了支持复杂的语法，在代码生成效率方面不免有所下降。为此，1995 年初在日本成立了 Embedded C 技术委员会，他们针对嵌入式应用制定了减小代码尺寸的 EC++标准。EC++保留了 C++的主要优点，提供对 C++的向上兼容性，并满足嵌入式系统设计的一些特殊要求。

C/C++/EC 引入嵌入式系统，使得嵌入式开发和个人计算机上的开发上的差别正在逐渐消除，软件工程的很多经验，方法乃至库函数都可以移植到嵌入式系统。在嵌入式开发中采用高级语言，还使得硬件开发和软件开发可以分工，从事嵌入式软件开发不再必须精通系统硬件和相应的汇编语言指令集了。

另一种高级语言，JAVA 的发展则具有戏剧性。JAVA 本来是为设备独立的嵌入式系统设计的，为了提高程序继承性的语言，但是目前基于 JAVA 的嵌入式开发工具代码生成长度要比嵌入式 C 编译工具差 10 倍以上。因此 EC++很可能将是未来的主流工具。

(3) 源程序模拟器

模拟器是在广泛使用的、人机接口完备的工作平台上，如小型机和 PC，通过软件手段模拟执行为某种嵌入式处理器内核编写的源程序测试工具。简单的模拟器可以通过指令解释方式逐条执行源程序，分配虚拟存储空间和外设，供程序员检查；高级的模拟器可以利用计算机的外部接口模拟出处理器的 I/O 电气信号。不同档次和功能模拟器工具价格差距很大。

模拟器软件独立于处理器硬件，一般与编译器集成在同一个环境中，是一种有效的源程序检验和测试工具。但值得注意的是，模拟器毕竟是以一种处理器模拟另一种处理器的运行，在指令执行时间、中断响应、定时器等方面很可能与实际处理器有相当的差别。另外它无法和 ICE 一样，仿真嵌入式系统在应用系统中的实际执行情况。

本论文的开发首先就是在源程序模拟器上进行开发和调试的。

(4) 实时多任务操作系统(Real Time Multi-tasking Operation System, RTOS)

实时多任务系统是嵌入式应用软件的基础和开发平台。目前在中国大多数嵌入式软件开发还是基于处理器直接编写，没有采用商品化的 RTOS，不能将系统软件和应用软件分开处理。RTOS 是一段嵌入在目标代码中的软件，用户的其他应用程序都建立在 RTOS 之上。不但如此，RTOS 还是一个可靠性和可信性很高的实时内核，将 CPU 时间、中断、I/O、定时器等资源都包装起来，留给用户一个标准的 API，并根据各个任务的优先级，合理地在不同任务之间分配 CPU 时间。

RTOS 是针对不同处理器优化设计的高效率实时多任务内核，优秀商品化的 RTOS 可以面对几十个系列的嵌入式处理器的 API 接口。这是 RTOS 基于设备独立

的应用程序开发基础。因此基于 RTOS 上的 C 语言程序具有极大的可移植性。据专家测算，优秀 RTOS 上夸处理器平台的程序移植之需要修改 1~5%的内容。在 RTOS 基础上可以编写出各种硬件驱动程序、专家库函数、行业库函数、产品库函数，和通用性应用程序一起，可以作为产品销售，促进行业内的知识产权交流。因此 RTOS 又是一个软件开发平台。

本论文在嵌入式 Linux 下设计和实现了 Email 功能。

1.2 TCP/IP 协议族

TCP/IP 是用于计算机通信的一组协议，我们通常称它为 TCP/IP 协议族。它是 70 年代中期美国国防部为其 ARPANET 广域网开发的网络体系结构和协议标准，以它为基础组建的 INTERNET 是目前国际上规模最大的计算机网络，正因为 INTERNET 的广泛使用，使得 TCP/IP 成了事实上的标准。之所以说 TCP/IP 是一个协议族，是因为 TCP/IP 协议[2]包括 TCP、IP、UDP、ICMP、RIP、TELNETFTP、SMTP、ARP、TFTP 等许多协议，这些协议一起称为 TCP/IP 协议。与 ISO 的 OSI 七层参考模型不同的是，TCP/IP 不是作为标准制定的，而是产生于 Internet 的研究和应用的实践中。虽然用 OSI 参考模型也可以描述 TCP/IP 协议，但是两者在细节上还是有一些差异。

1.2.1 TCP/IP 协议族分层结构

从协议分层模型方面来讲，TCP/IP 由五个层次组成：物理层、链路层、网络层、传输层、应用层。由于物理层是纯硬件这里不做详细讨论，剩下的四层的结构如图 1.1 中所示：

其中：

链路层是 TCP/IP 软件的最低层，负责接收 IP 数据报并通过网络发送之，或者从网络上接收物理帧，抽出 IP 数据报，交给 IP 层。

网络层 负责相邻计算机之间的通信。其功能包括三方面。

(1) 处理来自传输层的分组发送请求，收到请求后，将分组装入 IP 数据报，填充报头，选择去往信宿机的路径，然后将数据报发往适当的网络接口。

(2) 处理输入数据报：首先检查其合法性，然后进行寻径——假如该数据报已到达信宿机，则去掉报头，将剩下部分交给适当的传输协议；假如该数据报尚未到达信宿，则转发该数据报。

(3) 处理路径、流量控制、拥塞等问题。

传输层 提供应用程序间的通信。其功能包括：一、格式化信息流；二、提供

可靠传输。为实现后者，传输层协议规定接收端必须发回确认，并且假如分组丢失，必须重新发送。

应用层 向用户提供一组常用的应用程序，比如电子邮件、文件传输访问、远程登录等。远程登录 TELNET 使用 TELNET 协议提供在网络其它主机上注册的接口。TELNET 会话提供了基于字符的虚拟终端。文件传输访问 FTP 使用 FTP 协议来提供网络内机器间的文件拷贝功能。

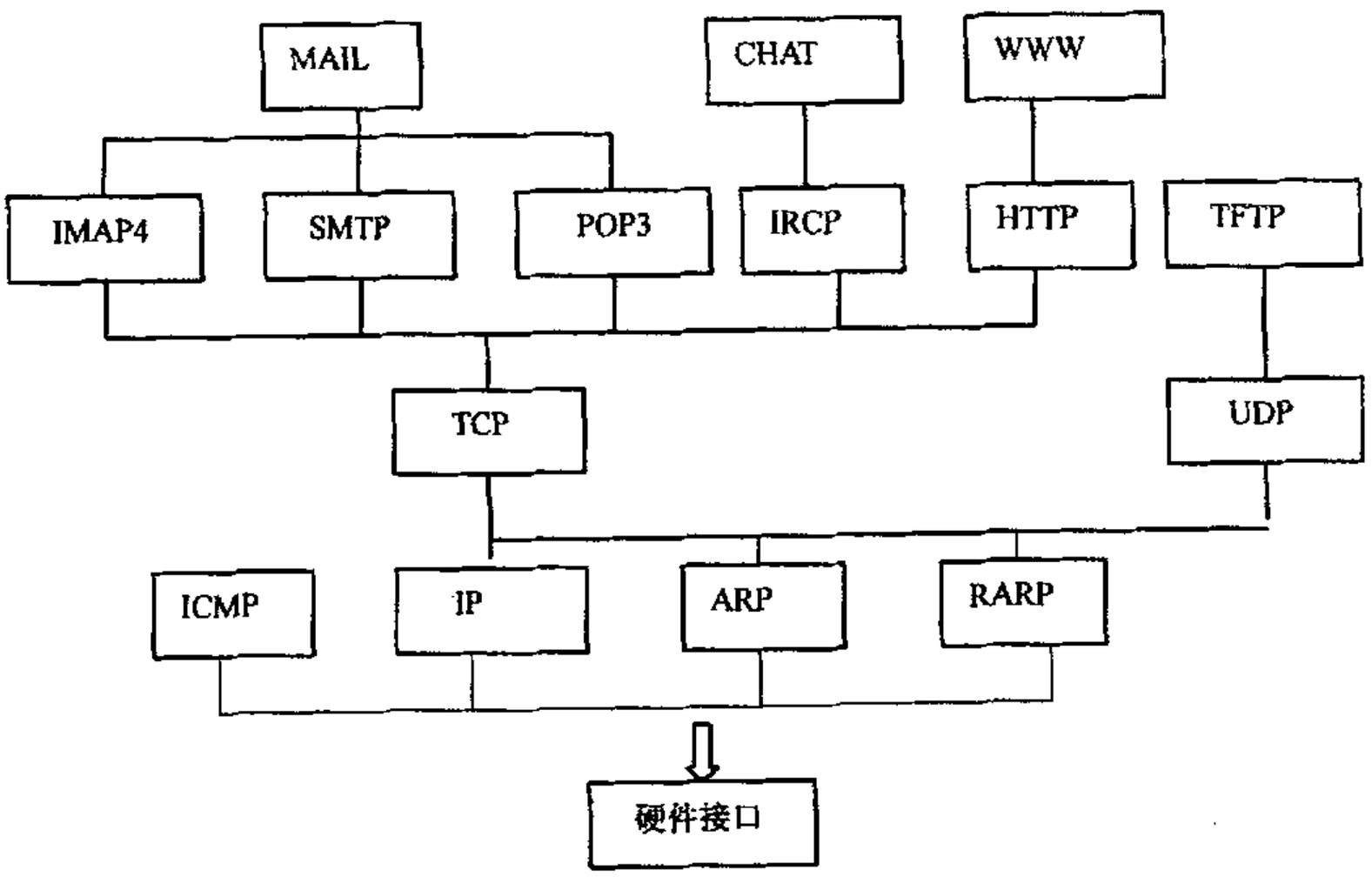

图 1.1 TCP/IP 协议族分层结构示意图[7]

Fig.1.1 Layer Structure of TCP/IP Suite [7]

1.2.2 TCP/IP 特点

TCP/IP 协议的核心部分是传输层的协议(TCP、UDP)，网络层协议(IP)和物理接口层，这三层通常是在操作系统内核中实现。因此用户一般不涉及。编程时，编程界面有两种形式：

(1) 是由内核心直接提供的系统调用；

(2) 使用以库函数方式提供的各种函数。

前者为核内实现，后者为核外实现。用户服务要通过核外的应用程序才能实现，所以要使用套接字(socket)来实现。

图 1.2 是 TCP/IP 协议核心与应用程序关系图。

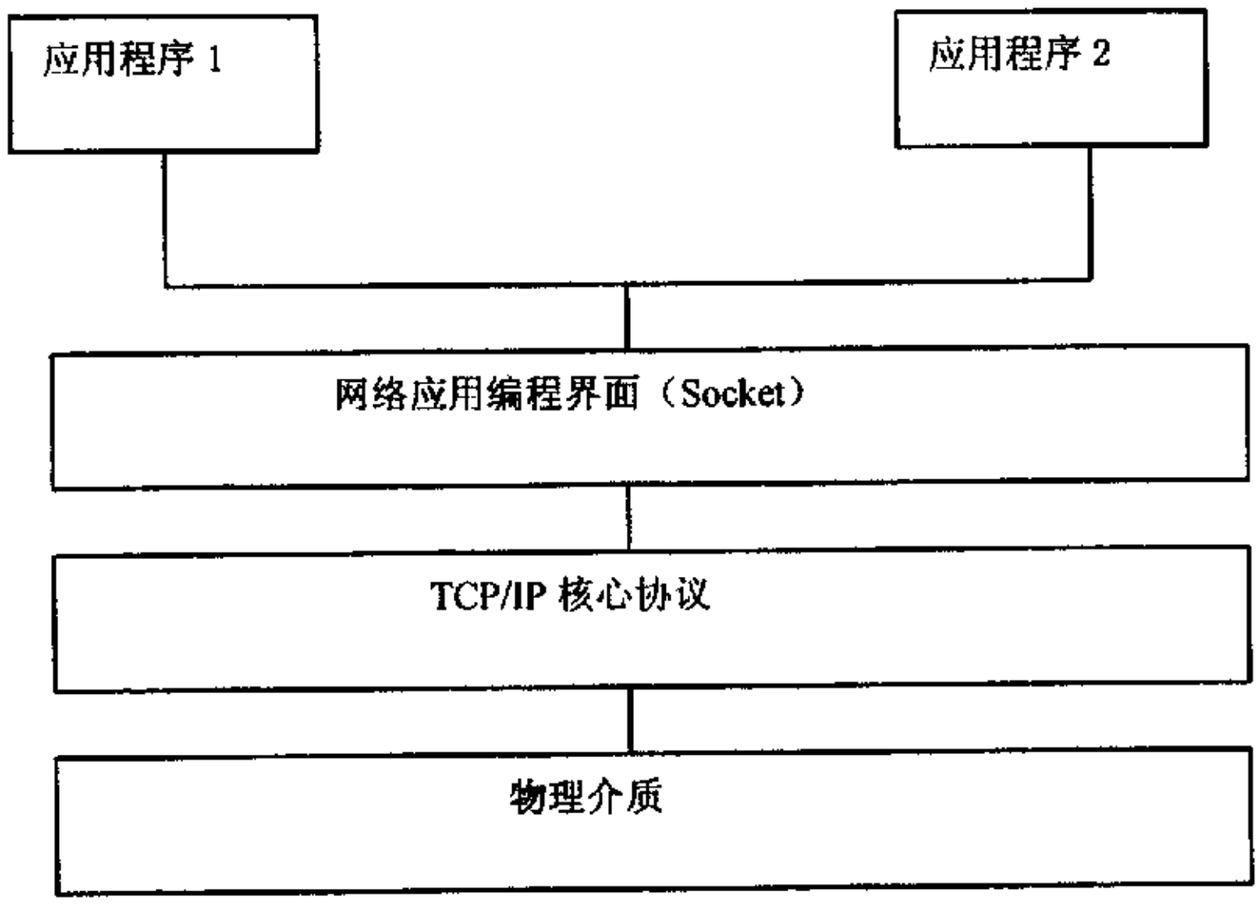

图 1.2 TCP/IP 协议与应用程序关系图[8]

Fig.1.2 Relation between the TCP/IP Protocols and the Applications [8]

1.2.3 套接字(Sockets)

套接字是网络通信的基本构件。套接字是可以被命名和寻址的通信端点，使用中的每一个套接字都有其类型和一个与之相连的进程。

套接字存在于通信区域中。通信区域也叫地址族，它是一个抽象概念，主要用于将通过套接字通信的进程的共有特性综合在一起。套接字通常只与同一区域中的套接字交换数据（也有可能跨越区域通信，但这只在执行了某种转换进程后才能实现）。Windows Sockets 只支持一个通信区域：网际域（AF-INET），这个域被网际协议族通信的进程使用。

套接字都具有类型，它是根据用户可见的通信特征进行分类的。应用程序被假定为只在同一类型的套接字间通信，不过只要依据的通信协议支持，也完全可以在不同类型的套接字间通信。Windows Sockets 版本 1.1 支持两种套接字：面向连接套接字（SOCK_STREAM）和无连接套接字（SOCK_DGRAM）。

在 Windows 环境性实现 Sockets 是指实现了 Windows Sockets 规范所描述的全部功能的一套软件。一般来说，在 Windows 下实现 Windows Sockets 功能都是通

过 DLL 实现的，并且很多实现是纯粹通过 DLL 实现的，因此，本文中的 Windows Sockets 实现的提法可以说等价于 Windows Sockets DLL。

第二章 系统开发环境介绍

本论文首先是在一个嵌入式设备的模拟器上实现电子邮件功能和在线聊天功能，然后把电子邮件功能移植到一个嵌入式 Linux 系统上去运行。因此本文所应用到的开发环境有两个：一是嵌入式系统模拟器，另一是在嵌入式 Linux。

2.1 嵌入式系统模拟器

2.1.1 嵌入式设备和模拟器之间的差异

图 2.1 是嵌入式设备的工作环境，底层是一个嵌入式操作系统，各应用程序是通过 MMI(MAN-MACHINE INTERFACE)程序实现，各个系统设备的消息发送（例如：按键、有新邮件等）都是通过此嵌入式操作系统的服务程序(service routine) 的系统调用实现的。

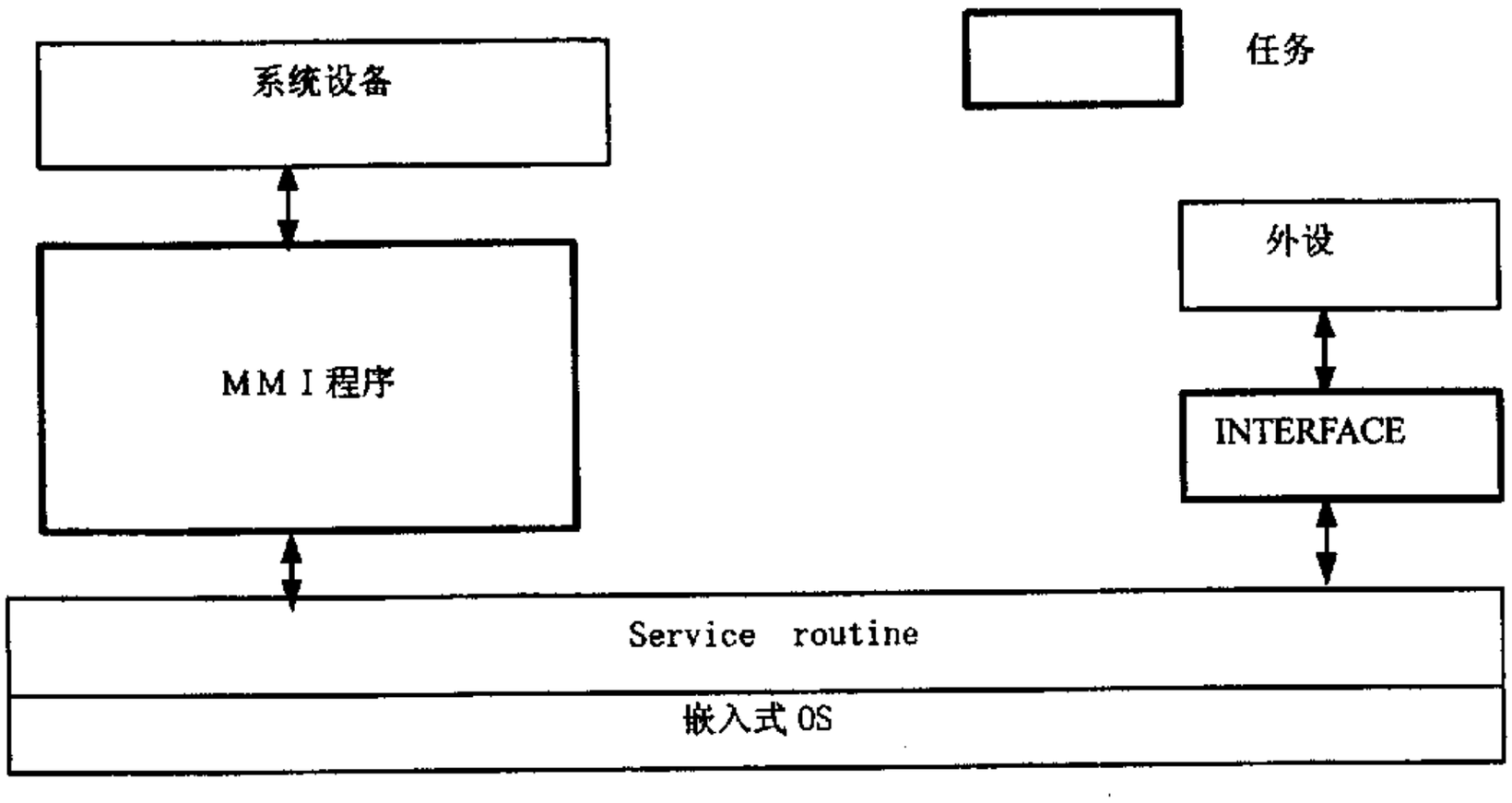

图 2.1．嵌入式设备上的操作环境

Fig2.1 Operating Environment on Embedded Device

虽说嵌入式设备的模拟器中的ＭＭＩ程序就是在 Windows 模拟开发嵌入式系统的应用程序，但是进行一些以下改变还是非常有必要的:

(1) 由于由嵌入式 OS 转变为 Windows OS，所以消息发送改变为 Windows 进程间的消息发送处理。

(2) 由于不存在外设，所以要对外围设备的动作进行模拟、并对外围设备状态的进行监控。

考虑上述两中改变后的环境见图 2.2。

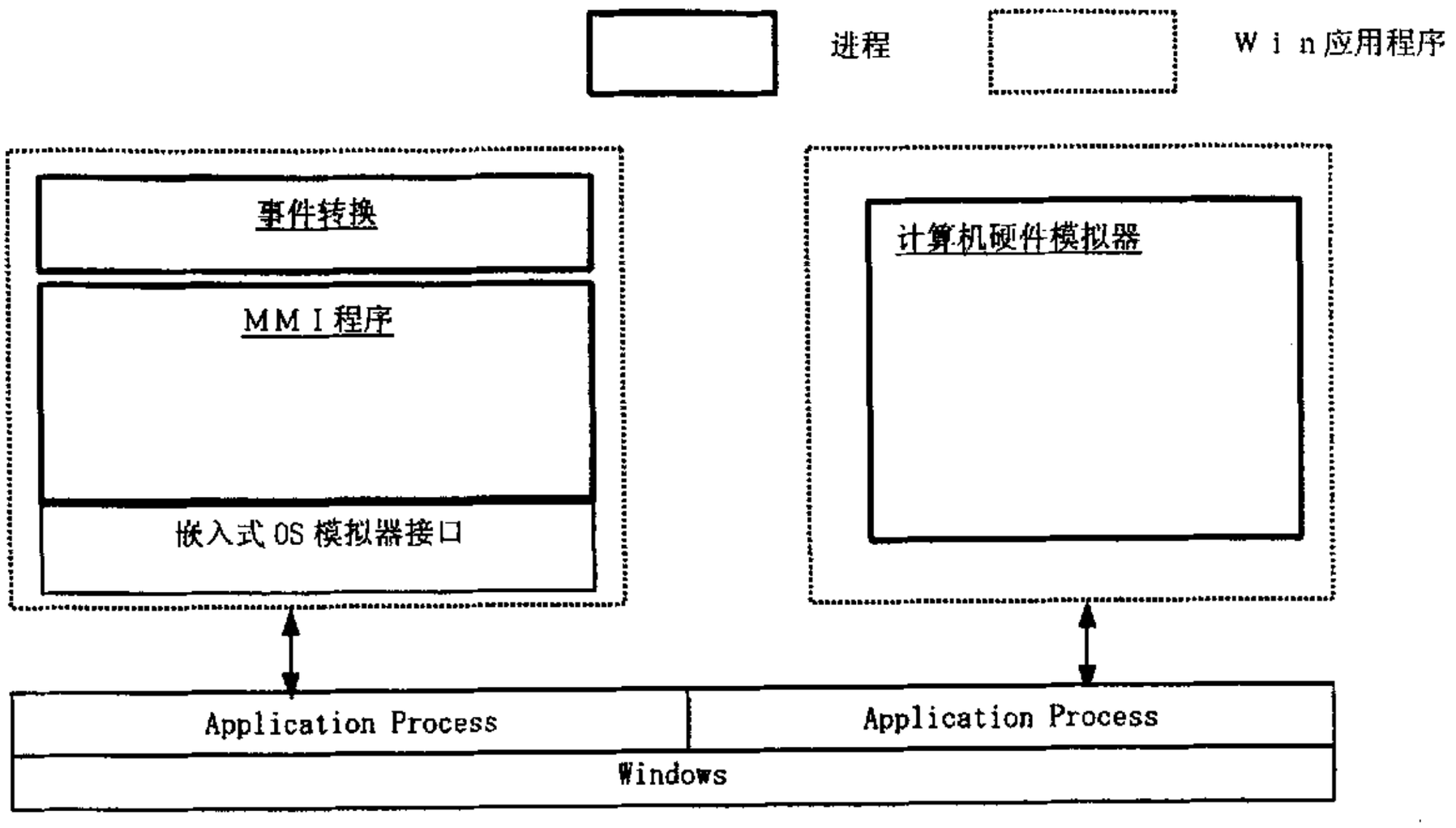

图 2.2 Windows 上操作环境

Fig2.2 Operating Environment on Windows

2.1.2 模拟器的构成

嵌入式设备模拟器是在 Windows OS 是利用多线程来实现的。由以下三个功能模块构成。

(1) 硬件模拟模块

硬件模拟模块有以下 3 个功能组成。

①硬件状态监控

②操作模拟

③应用程序状态转换模拟

(2) 事件转换模块

把硬件模拟模块中的各种事件转换成一个消息（事件）发送给 MMI 程序模块，并且使用一个线程来进行应用程序的管理。

(3) MMI（Man-Machine Interface）程序模块

嵌入式设备的实际动作在模拟器应用程序模拟的动作是通过动态连接库（dll）提供的。完成相应处理，最后给硬件模拟模块发送一个消息，使硬件模拟模块进行相应的操作。

这三个功能模块相互之间的关系见图 2.3。

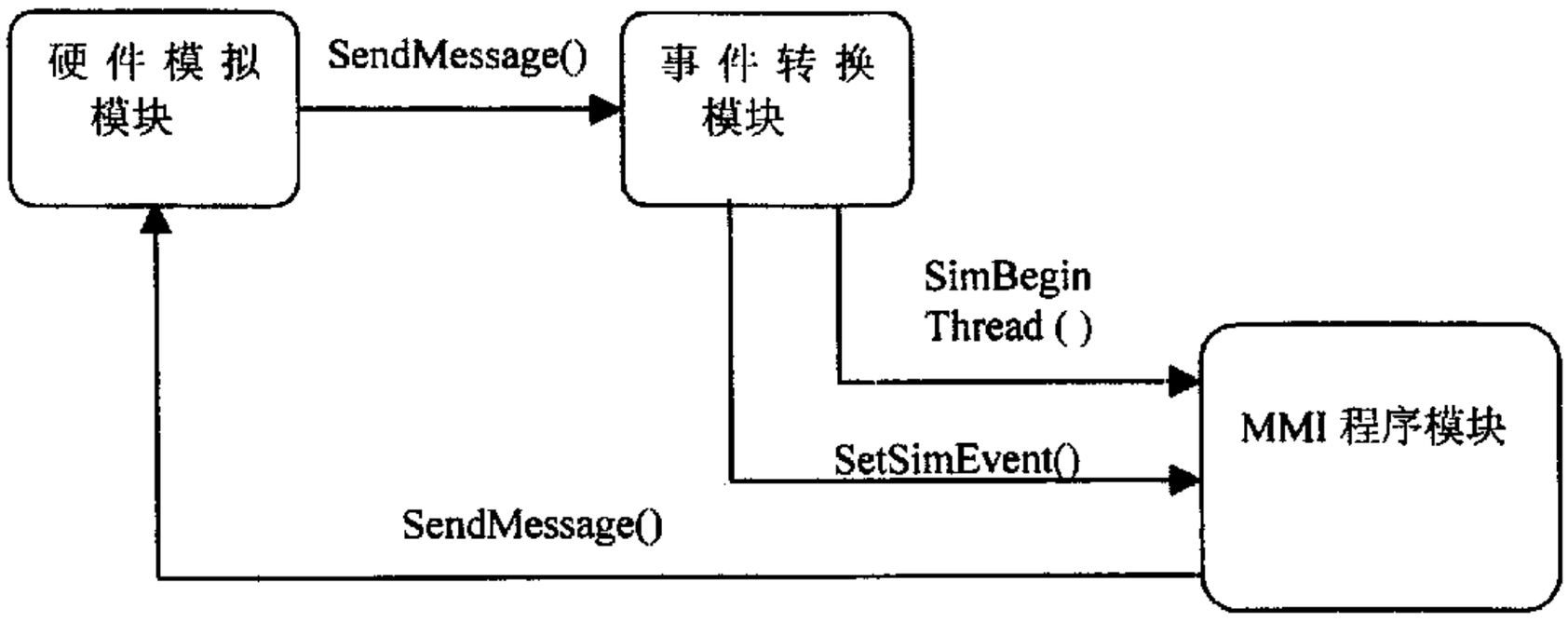

图 2.3 模拟器功能模块关联图

Fig.2.3 Relation among the Modules of Simulator

各个模块通信过程如下：

① 硬件模拟模块到事件转换模块：各个单独的应用程序使用 SendMessage( ) 函数进行通信。

② 事件转换模块到 MMI 程序模块 ：启动一个线程进行通信，并发送一个消息。

③ MMI 程序模块到硬件模拟模块：各个单独的应用程序使用 SendMessage( ) 函数进行通信。

根据以上分析，可以得到事件处理的时序图。图 2.4 是按键事件发生的时序图，其它事件的时序图和他类似。

MMI 程序模块是本嵌入式系统的核心部分，所有的 App 都是在此进行处理的。

这个函数的主要处理过程如下：

1）初始化处理：从电源 OFF 到电源 ON 时的处理过程

・Application List 初期化

・Emulation Device 初期化

・各 Task 初期化

2）事件取得

用 GetSimEvent 函数取得事件转换模块发送来的事件。

3）事件解析

根据 PowerON / OFF 的状态调用 ProcPowerOn / Off 进行事件解析。

4）实时控制

根据所解析的事件进行相应的处理。

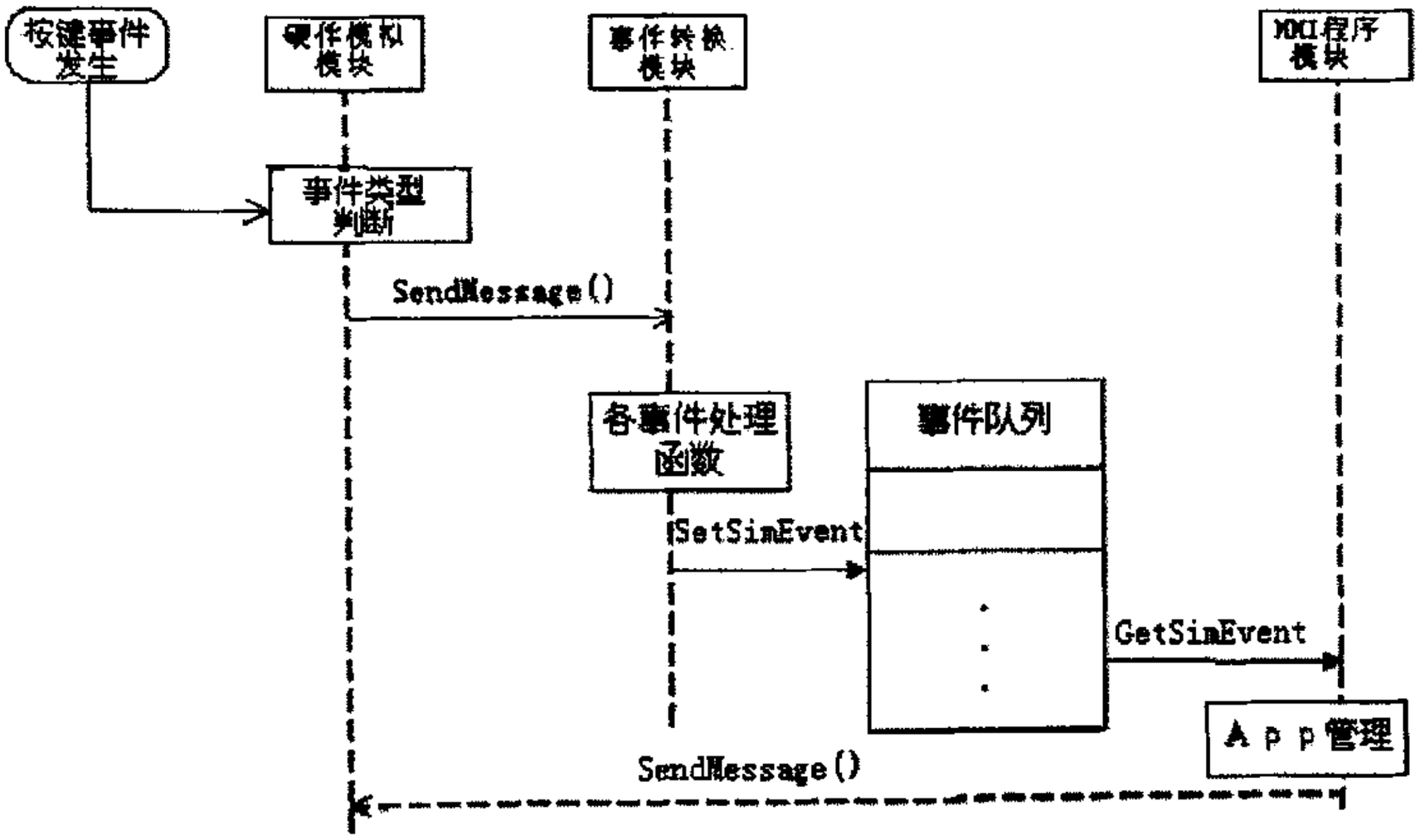

图 2.4　按键事件发生时序图

Fig.2.4 Sequence Diagram of Pressing Key Event

2.2 嵌入式 Linux

嵌入式 Linux[4]则是按照嵌入式操作系统的要求而设计的一种小型操作系统。由一个 Kernel（内核）及一些根据需要进行定制的系统模块组成。其 Kernel 很小，一般只有几百 KB 左右。即使加上其他必须的模块和应用程序，所需的存储空间也很小。它有多任务、多进程的系统特征，有些还具有实时性。一个小型的嵌入式 Linux 系统只需要引导程序、Linux 微内核、初始化进程 3 个基本元素。运行嵌入式 Linux 的 CPU 可以是 x86、Alpha、Sparc、MIPS、PPC 等。与这些芯片搭配的主板都很小，与一张 PCI 卡大小相当，有的甚至更小。嵌入式 Linux 所需的存储器不是软磁盘、硬盘、Zip 盘、CD-ROM、DVD 这些众所周知的常规存储器，它使用 Rom、CompactFlash、M-Systems 的 DiskOnChip、Sony 的 MemoryStick、IBM 的 MicroDrive 等体积极小——与主板上的 BIOS 大小相近，存储容量不太大的存储器。它的内存可以使用普通的内存，也可以使用专用的 RAM。

目前嵌入式 Linux 的应用非常的广泛，而且根据相关的调查和专家的分析，嵌入式 Linux 的应用会越来越多。是什么力量促使嵌入式 Linux 在这么短的时间就让包括一些专家在内的用户喜欢上它并使用它 的呢？嵌入式 Linux 究竟有什么

优点是别的嵌入式操作系统无法比拟的呢？经过一些简单的分析，我们就很容易得到嵌入式 Linux 的优点。这些优点是非常明显但是非常实际的。

与其他的嵌入式操作系统相比 Linux 的优势在于：

- Linux 源代码是开放的
- 强大的网络功能
- 灵活性
- 强大的开发者作后盾
- 强大的工具集[4]

第三章 嵌入式系统模拟器下电子邮件的设计和实现

3.1 基于 SMTP 和 POP3 协议的电子邮件功能的设计和实现

3.1.1 协议概述

电子邮件是使用最为广泛的应用层服务之一，许多用户第一次接触计算机网络就是发送电子邮件到远端网络节点或从远端网络节点接收电子邮件。TCP/IP 协议族分开提供邮件报文格式和邮件传输的标准。目前，大部分 Internet Email 系统都使用简单邮件传输协议（Simple Mail Transfer Protocol SMTP），SMTP 已经成为应用最广泛的上层协议之一。

SMTP 定义了一台计算机上的邮件系统如何将邮件传送到另一台计算机上。与大多数 TCP/IP 协议一样，SMTP 协议也是一种纯粹基于 C/S 计算机模型的应用层协议。SMTP 的协议模型包括一个发送器和一个接收器，两者都要访问一个文件系统进行信息存储。尽管 SMTP 并不提供绝对的端到端的可靠性的保障，但它仍然能够建立一个可以信赖的信息传递系统。因为它是基于 TCP 传输协议之上的，这也增加了它的可靠性。

电子邮件信息的传递可以划分为几个阶段，这几个阶段均由 SMTP 协议模型支持，如图 3.1 所示。从图中我们可以看到，邮件是以文件的形式存储的，对邮件的操作都可以用文件的操作来完成。邮件发送的具体过程可以分以下四个步骤完成：

第一步，用户向一个被称为用户代理的接口系统内提供输入信息，这个接口便于邮件信息的进入。

第二步， 信息被发送到 Sender-SMTP，它为该过程分配一个任意的端口号并与它的同层实体——Receiver-SMTP 通过 25 端口建立一个 TCP 连接。在此连接建立的过程中，接收者向发送者说明自己的身份。

第三步，发送者就可以使用 RFC822[8]种描述的格式发送邮件信息了。

最后，发送者发送终止连接消息，接收者对其进行确认。确认完毕后，TCP 连接就被释放。

到这步 ，整个的发送过程就已经完成。接收者就可以去接收此邮件了，当然在真正实现的过程中，还要考虑很多其它的因素。例如对于嵌入式上的邮件发送就要考虑发送的效率，就是发送的速度要尽量的快，占用的资源要尽量的少，还

有对于各个实时性很高的事件要及时的响应等等。

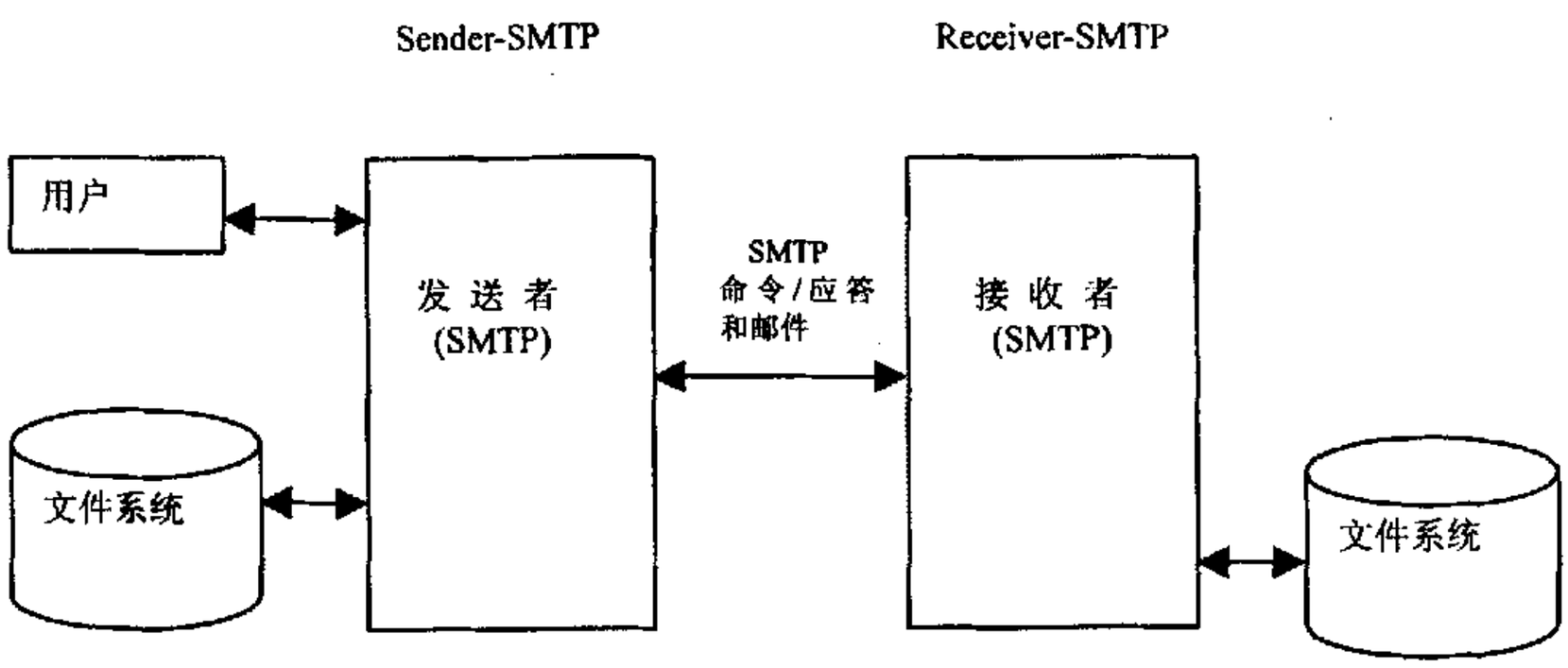

图 3.1 SMTP 协议模型示意图

Fig3.1 Model of SMTP Protocol

前面介绍的 SMTP 传输方案暗示服务器必须在任何时候做好接收电子邮件的准备。一旦用户输入邮件，服务器就试图将其发送出去。如果服务器运行在具有永久互联网连接的计算机上，则能够做到这一点。但对于并非一直连接互联网上的计算机，则无法达到目的。考虑到一般的用户，要永久连接到互联网上是不现实的。那样，对于一个没有永远连接的用户是怎样接收电子邮件的呢？

这个问题的答案在于一个两阶段交付过程。在第一阶段，在具有永久 Internet 连接的计算机上为每个用户分配一个邮箱。这台计算机运行一个 SMTP 服务器，SMTP 服务器一直准备着接收电子邮件。在第二阶段，用户建立一个连接，然后运行一个从永久邮箱检索邮件的协议。这个协议把邮件传输到用户使用的计算机，在这台计算机上阅读邮件。

有两个协议可允许远程用户从永久邮箱检索邮件。邮局协议(Post Office Protocol POP3)和网络消息访问协议(Internet Message Access Protocol IMAP4)。 虽然，您可以使用两种协议的任何一种，但是，它们的能力和使用方法是差别很大的，下面论文将介绍 POP3 协议，IMAP4 协议和这两个协议之间的区别将在下部分介绍。

POP3 采用的是一种离线的模式，下面我们来看它是怎样工作的。当您打开邮件程序时，它会先检查是否已存在网络连接，如果没有，它就会建立一个连接通道 (通常是 PPP - Point-to-Point Protocol，即点对点协议连接)。然后，它会向 POP3 服务器发出一个登录请求，并把您的用户名和密码传给服务器。如果服务器验证

合法，邮件程序就会把服务器上的邮件下载到本地，并删除它们。这时，您就可以断开连接，阅读这些邮件了。图 3.2 是 POP3 协议模型示意图。

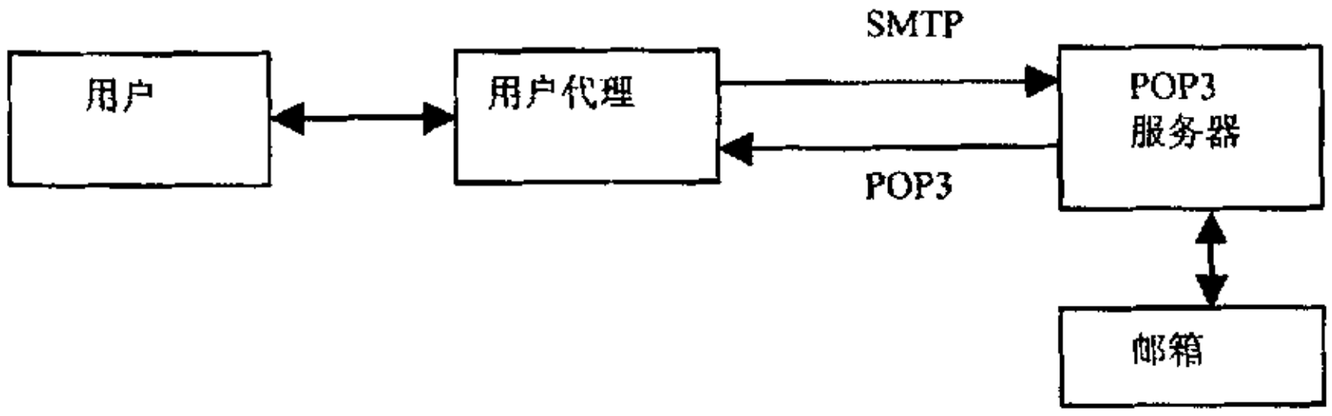

图 3.2　POP3 协议模型示意图

Fig.3.2 Model of POP3 Protocol

初始时，服务器通过侦听 TCP 端口 110 开始 POP3 服务。当客户主机需要使用服务时，它将与服务器主机建立 TCP 连接。当连接建立后，POP3 发送确认消息。客户和 POP3 服务器相互（分别）交换命令和响应，这一过程一直要持续到连接终止。

POP3 命令由一个命令和一些参数组成。所有命令以一个 CRLF 对结束。命令和参数由可打印的 ASCII 字符组成，它们之间由空格间隔。命令一般是三到四个字母，每个参数却可达 40 个字符长。

POP3 响应由一个状态码和一个可能跟有附加信息的命令组成。所有响应也是由 CRLF 对结束。现在有两种状态码，"确定" ("+OK")和"失败" ("-ERR")。

对于特定命令的响应是由许多字符组成的。在这些情况中，下面一一表述：在发送第一行响应和一个 CRLF 之后，任何的附加信息行发送，他们也由 CRLF 对结束。当所有信息发送结束时，发送最后一行，包括一个结束字符（十进制码 46，也就是"."）和一个 CRLF 对。如果信息中的任何一行以结束字符开始，此行就是通过在那一行预先装入结束而进行字符填充的。因此，多行响应由五个 CRLF.CRLF 结束。当检测多行响应时，客户检测以确认此行是否以结束字符开始。如果是的，而且其后的字符不是 CRLF，此行的第一个字符（结束字符）将被抛弃；如果其后紧跟 CRLF，从 POP 服务器来的响应终止，包括.CRLF 的行也不被认为是多行响应的一部分了。

在生命周期中，POP3 会话有几个不同的状态。一旦 TCP 连接被打开，而且 POP3 服务器发送了确认信息，此过程就进入了"确认"状态。在此状态中，客户必须向 POP3 服务器确认自己是其的客户。一旦确认成功，服务器就获取与客户邮件

相关的资源，此时这一过程进入了"操作"状态。在此状态中，客户提出服务，当客户发出 QUIT 命令时，此过程进入了"更新"状态。在此状态中，POP3 服务器释放在"操作"状态中取得的资源，并发送消息，终止连接。

POP3 服务器可以拥有一个自动退出登录的记时器。此记时器必须至少可以记录 10 分钟。这样从客户发送的消息才可能刷新此记时器。当记时器失效时，POP3 会话并不进入"更新"状态，而是关闭 TCP 连接，而且不删除任何消息，不向客户发送任何响应。

3.1.2 功能的设计和实现

3.1.2.1 功能的设计

在模拟器系统中，模拟了整个嵌入式系统的状态和操作。Email 功能是这个嵌入式系统的一个基本的功能。图 3.3 是此模拟器系统中的 Email 功能的构成图。从图中我们可以看到，Email 功能可以从 MAIN, MENU, EDIT, MAILBOX, 等模块中启动。然后通过发送接收模块对 Email 的发送接收进行控制，最后是解析邮件发送和接收协议与服务器进行通信。整个 Email 功能的核心部分就是对邮件发送和接收过程中所使用 TCP/IP 应用层协议的解析，只要成功解析了各个协议，Email 发送和接收的功能也就相应的完成了。最后的通信过程只不过是利用 Socket 进行服务器端和客户机端进行数据交换的过程。其间的数据交互的过程与协议无关，这样实现的好处就在于解析过程和通信过程完全的分开，模块化就非常的好，相互之间的影响非常的少，也就是说，解析过程中的问题不会影响通信过程，反之亦然。这对程序的编码和测试都是非常有用的。

从上面分析可以看到，Email 功能实现的关键点就是在协议解析上，所有的数据都是在这个部分处理的，而且又要与服务器进行相关的通信，如果把这两部分功能放在一起处理的话，程序的结构将非常复杂，而且也没有通用性。所以非常有必要把这两部分分别处理，一个模块专门用于协议数据的解析，而另一部分用于与服务器通信。这样在协议解析过程中就不用关心当前的网络状态，而在服务器通信过程中也只管接收数据而不用在乎是什么数据，错误信息和正确信息处理在这里处理是完全一样的。解析过程和通信过程分开处理可以保持程序的灵活性，而且代码也比较容易理解。

邮件发送过程的设计是按照 RFC2554[6]的描述完成的。为了更好地说明邮件的发送过程，可使用发送过程的时序图（参照图 3.4）来说明服务器和客户机交互的过程。

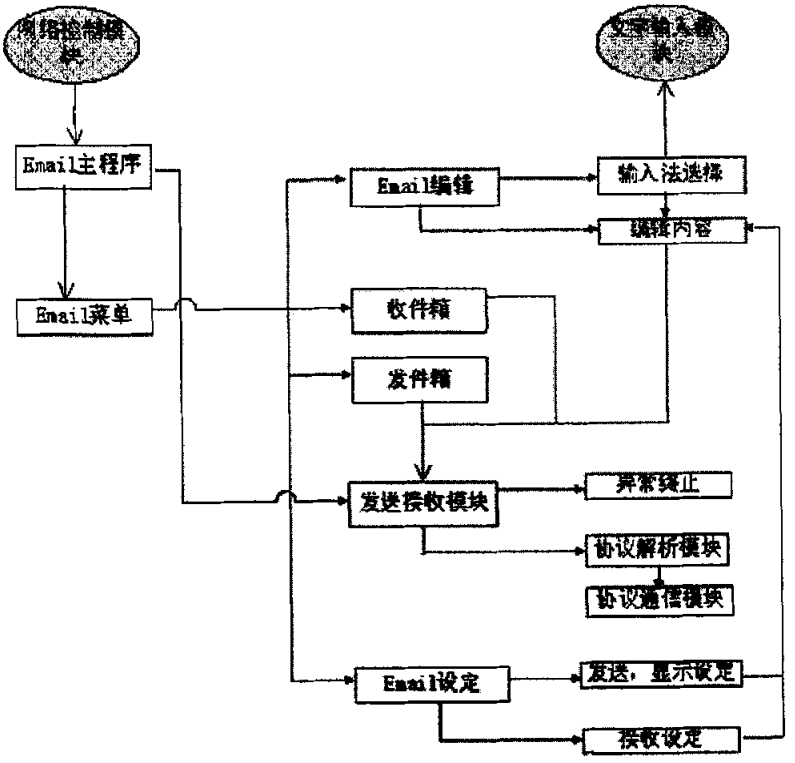

图3.3 Email 构成图

Fig. 3.3 Configuraton of Email

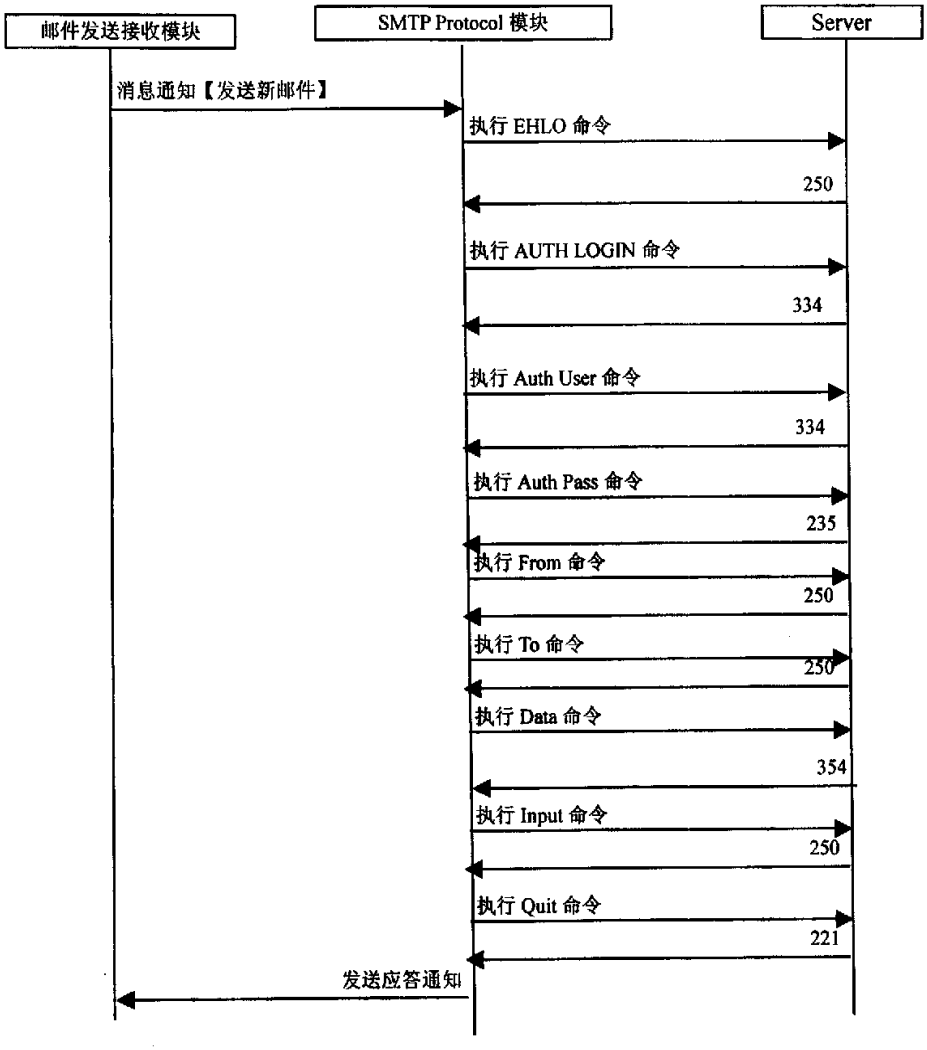

图 3.4 SMTP 发送邮件顺序图

Fig.3.4 Sequence Diagram of Sending Email Using SMTP Protocol

基于 POP3 协议邮件接收功能的设计的过程也是非常的有规律的，依照 RFC1939[13]所描述的步骤，并依据服务器的相关设置，最终完成基于 POP3 协议接收邮件的功能的设计。整个邮件的接收的顺序图如图 3.5 所示。

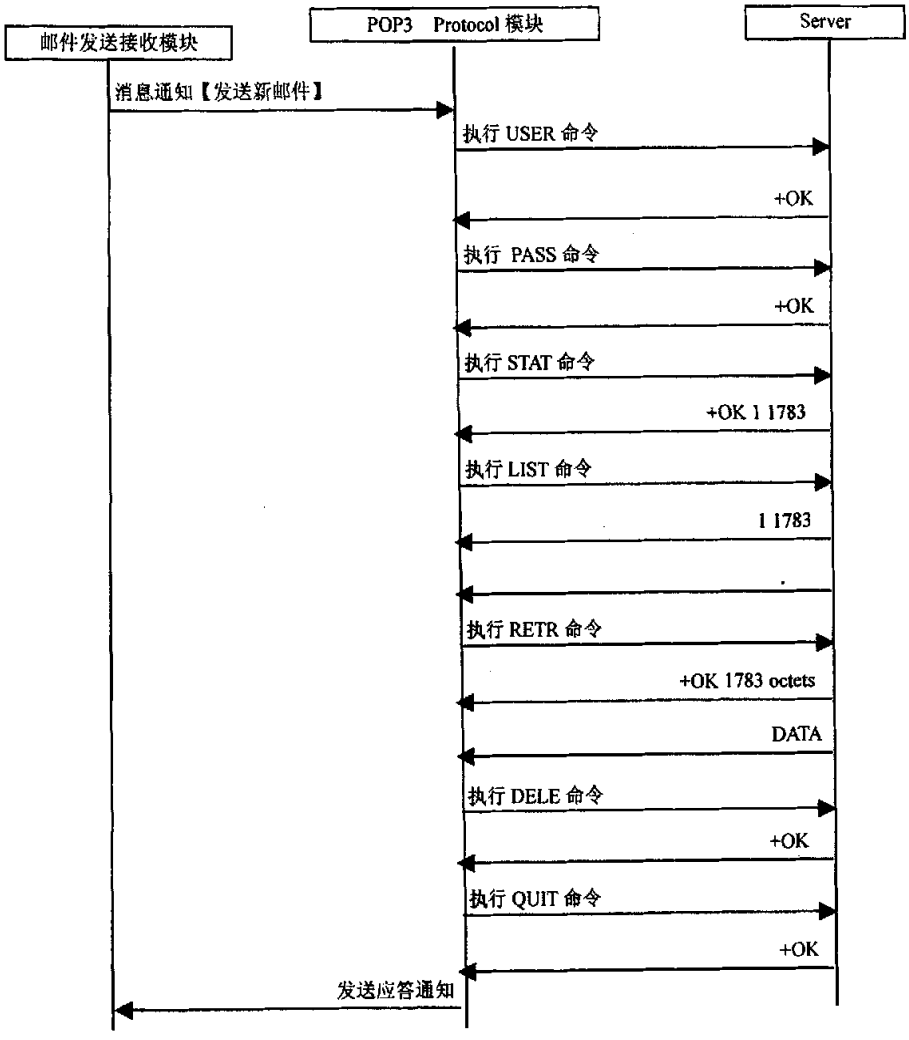

图 3.5 POP3 接收邮件顺序图

Fig3.5 Sequence of Receiving Email Using POP3 Protocol

3.1.2.2 功能的实现过程

SMTP 协议的实现比较规范，下面就是本模拟器根据 SMTP 协议实现邮件发

送的全部过程(假设本模拟器连接的 SMTP 服务器为 smtp.neusoft.com)：

S：220 smtp.neusoft.com – Server ESMTP (Neusoft Mail Server)
C：EHLO
S：250-smtp.neusoft.com
250-8BITMIME
250-PIPELINING
250-DSN
250-XDFLG
250-ENHANCEDSTATUSCODES
250-HELP
250-TURN
250-XADR
250-XLOOP C8C679C439667900E4BBFFC34FD2744
250-AUTH LOGIN PLAIN
250-AUTH=LOGIN
250-ETRN
250-RELAY
250-SIZE 0
C：AUTH LOGIN
S：334 VXNIcm5hbWU6
（注：username 的 base64 编码，提示用户输入用户名）
C：emhvdWpn (注：用户名的 base64 编码)
S：334 UGFzc3dvcmQ6
（注：password 的 base64 编码，提示用户输入用户密码）
C：amlhbmd4aWppbnhpc2h1eGk=（注：用户密码的 base64 编码）
S：235 2.7.0 LOGIN authentication successful
C：MAIL FROM：<yamamo@kkf.com>
S：250.2.5.0 Address OK.
C：RCPT TO：<zhoujg@neusoft.com>
S：250 2.1.5 zhoujg@neusfot.com OK
C：DATA
S：354 Enter mail, end with a single “. ”
C：This is a mail for testing
C：　.
S：250 2.5.0 ok.

C：QUIT

S：221 2.3.0 Bye received．Goodbye.

要完成发送邮件的功能，第一个任务就是要创建一个套接字，并试图根据邮件服务器定义的参数连接到服务器上。

为了进行连接，必须首先得到服务器的 IP 地址。邮件服务器参数可以包括用点分割的字符串 IP 地址或者完全合格的域名，例如：

“192.168.1.1”　　点分割的 IP 地址

“mail.163.com”　　完全合格的域名

首先假定字符串是点分割的 IP 地址，并且使用 inet_addr 函数来转换字符串到 32 位的 IP 地址。inet_addr 函数将返回-1，表示条件错误。这个错误表明提供的字符串不是一个有效的 IP 地址字符串。在这种情况下，可以假定提供的参数是一个完全合格的域名，同时使用 gethostbyname 函数来翻译域名成为 IP 地址。这个函数可以与配置好的域名服务器进行通信，解析地址，从而得到 32 位的 IP 地址。这里介绍的构造过程是一种十分有用的模式，十分容易的解决了域名和 IP 地址的问题。

一旦有了 IP 地址，就可以使用 connect 函数连接到服务器。现在就有一个连接邮件服务器和客户机的套接字。只要连接的是 SMTP 服务器，这个套接字上面的所有事务都是使用 SMTP 协议。

已经知道 SMTP 是一种命令应答协议，在实现过程中要发出命令，服务器以一串数字型应答码进行应答。使用这种方式，可以创建会话功能为模式化的 SMTP 事务。会话过程需要三个参数：套接字，命令字符行，应答码字符串。命令字符行就是客户机发送给 SMTP 服务器的命令，应答码字符串就是由服务器发出的表示成功以否的数字字符串。在有些情况下，命令行无法发出，或者得到了不是所期望的应答码。这时候就需要相关的处理。例如，当客户机第一次和 SMTP 服务器连接时，即可以得到来自服务器的简单的致意报文。这个字符串包括一个数字型应答码 220，表示客户机已经连接到有效的服务器上面。然而，如果命令没有发出，或者得到不期望的应答。就出错了。在客户机发送报文是，服务器是不做应答的，这个过程一直持续到邮件接收完成。

下面是实现 POP3 协议的过程中，其实现过程是完全遵照 RFC1939[13]的来实现，它与服务器会话的过程如此下：

```
S: +OK Messaging Multiplexor ( iPlanet Messaging Server 5.2)
C: USER zhoujg
S: +OK password required for user zhoujg aging Server 5.2
C: PASS ************
```

```
S: +OK Maildrop ready
C: STAT
S: +OK 1 1788
C: LIST
S: +OK Mailbox scan listing follows
S: 1 1788
S: .
C: RETR 1
S: +OK 1783 octets
Return-path: <zhoujg@neusoft.com>
Received: from e501.neusoft.com (e500.neusoft.com [202.107.117.25])
    by mail.neusoft.com (iPlanet Messaging Server 5.2 Patch 1 (built Aug 19 2002))
    with SMTP id <0HLS000UAS4A9A@mail.neusoft.com> for
zhoujg@ims-ms-daemon
    (ORCPT zhoujg@neusoft.com); Fri, 26 Sep 2003 09:28:58 +0800 (CST)
Received: from mail.neusoft.com(202.107.117.28) by e501.neusoft.com via csmap
    id 15943; Fri, 26 Sep 2003 09:32:34 +0800 (CST)
Received: from zjg ([192.168.31.65])
    by smtp.neusoft.com (iPlanet Messaging Server 5.2 Patch 1 (built Aug 19 2002))
    with ESMTPA id <0HLS003BUS4A2W@smtp.neusoft.com> for
zhoujg@neusoft.com; Fri,
    26 Sep 2003 09:28:58 +0800 (CST)
Date: Fri, 26 Sep 2003 09:30:45 +0800
From: zhoujg <zhoujg@neusoft.com>
Subject: test
To: zhoujg <zhoujg@neusoft.com>
Message-id: <002c01c383cd$cf1155e0$411fa8c0@zjg>
MIME-version: 1.0
X-MIMEOLE: Produced By Microsoft MimeOLE V6.00.2600.0000
X-Mailer: Microsoft Outlook Express 6.00.2600.0000
Content-type: multipart/alternative;
    boundary="Boundary_(ID_uqy1LFD7MMeAZa1XcwgKCA)"
X-Priority: 3
X-MSMail-priority: Normal

This is a multi-part message in MIME format.

--Boundary_(ID_uqy1LFD7MMeAZa1XcwgKCA)
```

```
Content-type: text/plain; charset=gb2312
Content-transfer-encoding: 7BIT

This email is just for testing

--Boundary_(ID_uqy1LFD7MMeAZa1XcwgKCA)
Content-type: text/html; charset=gb2312
Content-transfer-encoding: 7BIT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD>
<META http-equiv=Content-Type content="text/html; charset=gb2312">
<META content="MSHTML 6.00.2600.0" name=GENERATOR>
<STYLE></STYLE>
</HEAD>
<BODY bgColor=#ffffff>
<DIV>This email is just for test</DIV></BODY></HTML>

--Boundary_(ID_uqy1LFD7MMeAZa1XcwgKCA)--

.
C: DELE 1
S: +OK Message delete
C: QUIT
S: +OK
```

现在来分析一下这个对话过程，以便理解 POP3 协议是如何进行邮件传输的。首先，连接服务器 110 端口，当服务器存在的时候，这个 POP3 服务器立即发送以状态码(+OK)开头的致意信息，表示已经连接上 POP3 服务器，并且一切正常。一旦开始连接，便进入了授权状态。任何人不可以执行任何命令，直接从授权状态转到事务状态。这种转换是在使用命令 USER 和 PASS 向服务器认证了用户之后实现的。这两个命令为服务器识别用户名和口令。每个命令后需要跟上一个状态码和人们可读的文本。

在 PASS 命令被接受并且经过服务器的认证之后，会应答一个状态码(+OK)和服务器有多少可用报文的指示符。然后执行一个冗余 STAT 命令，告诉客户机有多少可用报文，以及这些报文总计多少字节。

使用 LIST 命令，客户机可以得到更多的有关邮件内容的详细信息。这个命令提供

了可用报文的清单以及每个报文的字节数。如果希望基于大小来使某些报文无效，这个功能就十分有用。应考虑到嵌入式系统具有有限数量的存储器，如果邮件太大，超过一定的标准，就可以立即删除这个报文，而不必去下载它。LIST 命令可以得到两个值纪录的邮件列表：邮件指数（邮件的 ID）和以字节计的大小。表示列表结束的标志为使用一个符号“.”单独占用一行。回想 SMTP 的讨论，这与邮件结束指示符相同。

获取一个邮件的内容使用的是 RETR 命令。使用 RETR 命令进行下载，客户机就需要指定邮件的数字形式的 ID，POP3 服务器首先回应的是一个状态指示符(此处为+OK)，然后就是报文的正文。请注意，状态指示符后面是需要下载的邮件的大小。这个在动态系统中十分有用，在这种动态系统中，被填充的缓冲器是动态产生的。然后接下来的部分就是完整的邮件的内容，后面跟着一个邮件结束指示符(CR/LF/ ‘.’ /CR/LF)。一旦收到这个指示符，客户机就是通知服务器返回命令模式，等待指令。

接下来的命令 DELE 用来删除服务器上的报文。请注意这个操作要进行删除，而实际上还没有被删除。直到从事务状态进入到更新状态，邮件才会被真正的删除。

最后就是使用 QUIT 命令结束会话，这时通知服务器进行更新，也就是进入到更新状态。这样，服务器就开始删除有删除标记的邮件，并且关闭套接字，响应最后一个成功指示符。

客户机发送服务器所期待的命令时，要以一个结束标记（回车换行）来结束。

3.1.2.3 实现过程中的关键问题及其解决方案

（1）、 身份验证

① SMTP 身份验证[6]

随着现在电子商务的发展，邮件传输的保密性要求也就越来越高，为了限制非本系统的正式用户利用邮件服务器散发垃圾邮件或进行其他不当行为，国内各大免费/收费邮箱提供商纷纷升级到 ESMTP 方式，开设了安全认证服务。在 ESMTP 服务器上，发送邮件需要对用户的身份进行验证。与传统的 SMTP 方式相比，多了一道用户身份的验证手续，验证之后的邮件发送过程与传统的 SMTP 方式一致。

几乎所有的 ESMTP 服务器都继承了 POP3 服务器的账号和密码设置体系，也就是说收发邮件用相同的账号和密码。当然，也可以用不同的账号和密码，但那样无论是电子邮件服务提供商的维护还是用户的使用都会很麻烦，故而很少采用。

ESMTP 身份验证的机制有很多种，最常见的是 LOGIN 机制，类似于 POP3

的身份验证方式，即分两步输入账号和密码。在所有的验证机制中，信息全部采用 Base64 编码。

SMTP 的认证功能主要是增加了 AUTH 命令。AUTH 命令有多种用法，而且有多种认证机制。AUTH 支持的认证机制主要有 LOGIN，CRAM-MD5[9]等。LOGIN 应该是大多数免费邮件服务器都支持的。认证机制一般只在真正发送邮件之前进行，而且只需要执行一次。当认证成功后，即可按原来正常的处理发送邮件。原理是口令-应答(Challenge-Response)，即由服务器发送命令要求客户端回答，客户端根据服务器发送信息进行回答，如果应答通过了，则认证成功，即可继续处理。下面对这两种制作一个简单介绍。S:表示服务器返回，C:表示客户端发送。

(a) LOGIN

它应该比较简单。口令-应答过程如下：

```
1  C: AUTH LOGIN
2  S: 334 dXNlcm5hbWU6
3  C: dXNlcm5hbWU6
4  S: 334 cGFzc3dvcmQ6
5  C: cGFzc3dvcmQ6
6  S: 235 Authentication successful。
```

1 为客户端向服务器发送认证指令。

2 服务端返回 base64 编码字符串，成功码为 334。编码字符串解码后为“username:”，说明要求客户端发送用户名。

3 客户端发送用 base64 编码的用户名，此处为“username:”。

4 服务端返回 base64 编码字符串，成功码为 334。编码字符串解码后为“password:”，说明要求客户端发送用户口令。

5 客户端发送用 base64 编码的口令，此处为“password:”。

6 成功后，服务端返回码为 235，表示认证成功可以发送邮件了。

对于 LOGIN 方式认证，其实就是将用户名与口令用 base64 进行编码，根据服务器的要求，分别发出即可。由于 base64 是一种公共的编码标准，也起不到太大的保护作用。

(b) CRAM-MD5 机制

关于 CRAM-MD5 的机制可以参考 RFC 2195[7]规范，这里不详细说明了。主要就是通过口令-回答机制，由服务端发出一个信息串，这个由随机数，时间戳，服务器地址构成，并且用 base64 编码。客户端收到后，发送一个由用户名，加一个空格，再加一个要构成的串，并用 base64 编码。摘要是通过 MD5 算法求出。这

种机制要求服务端与客户端有相同的加密串。当客户端发送摘要后，服务器对其合法性进行验证，成功后，返回235。

如何得知邮件服务器支持什么认证？在SMTP的RFC 821[11]中，在与邮件服务器连接成功后，第一个命令一般是"HELO"。但是在支持认证的邮件服务器中，第一个命令应改为"EHLO"。在命令成功后。

```
EHLO hello
250-smtp.neusoft.com
250-PIPELINING
250-SIZE 10240000
250-ETRN
250-AUTH LOGIN
250 8BITMIME
```

② POP3身份验证[7]

POP3协议接收邮件时的用户和密码的设定了，因为这个设定的过程与接收协议设定的过程非常相似。由于此时的数据是不确定的，所以不能使用枚举型数据，因此定义了一个两个字符型数组

char　Pop3Username[USERNAME_MAX+1]

char　Pop3Password[PASSWORD_MAX+1]，

用来存储这些信息。同样是定义了一些API函数供其它模块的调用：

ApiEmailLclSettingSetPOP3UserName(char *username)

ApiEmailLclSettingGetPOP3UserName(char *buf, int bufsize)

ApiEmailLclSettingSetPOP3UserPassword(char *userpassword)

ApiEmailLclSettingGetPOP3UserPassword(char *buf, int bufsize)

其中函数 ApiEmailLclSettingSetPOP3UserName(char *username) 和函数ApiEmailLclSettingSetPOP3UserPassword(char *userpassword)是在Email账户设定模块中调用，而调用函数 ApiEmailLclSettingGetPOP3UserName(char *buf, int bufsize)和函数ApiEmailLclSettingGetPOP3UserPassword(char *buf, int bufsize)的情况有：

(a) 初始化模块，从DB中得到相关设定。

(b) 协议解析模块，得到相关数据，并向服务器发送，以前得到验证。

(c) 菜单显示模块，显示当前的账户信息，并可以修改。

（2）邮件信息的存储，发送和接收

SMTP的消息格式非常直观，客户端和服务器之间的消息由可读的ASCII文本组成。

RFC822[8]定义了一个通过电子邮件发送文本消息的格式。它是基于 Internet

的文本邮件消息的标准，现在仍然在使用。在 RFC822 中，将消息定义为信封和消息正文组成，信封包括完成传递和投递所需要的所有信息，邮件正文部分包括发送给接收方的对象组成。RFC822 只适用于正文部分。但是正文标准包含了一组被邮件系统创建邮件头部时用到的报文字段，而且此标准可以便于程序对这种信息的读取。

与 RFC822 相一致的消息格式非常简单。消息由一些报头行（头部）和其后没有限制的文本（正文）构成。 头部和正文之间由一个空白行分割。头部中包括几个域。头部中有许多域是可选的，由本地主机的实际应用决定。其中最常用的域是：From、To、Subject 和 Date。

下面这个头部是本系统中所使用的头部。

```
Date: Fri, 26 Sep 2003 09:30:45 +0800
From: zhoujg <zhoujg@neusoft.com>
Subject: Hello
To: zhoujg <zhoujg@neusoft.com>
Message-id: <002c01c383cd$cf1155e0$411fa8c0@zjg>
MIME-version: 1.0
X-MIMEOLE: Produced By Microsoft MimeOLE V6.00.2600.0000
X-Mailer: Microsoft Outlook Express 6.00.2600.0000
Content-type: multipart/alternative;
boundary="Boundary_(ID_uqy1LFD7MMeAZa1XcwgKCA)"
X-Priority: 3
X-MSMail-priority: Normal
```

Email 只能传送 ASCII 码格式的文字信息，ASCII 码是 7 位代码，非 ASCII 码格式的文件在传送过程中就需要，先编成 7 位的 ASCII 代码，然后才能通过 E—mail 进行传送；如果不经过编码，则在传送过程中会因为 ASCII 码 7 位的限制而被分解，分解之后只会让收信人看到一堆杂乱的 ASCII 字符。经过编码后的文件，在传送过程中可顺利传送，不会有“被截掉一位”的危险。但是收信方必须具有相应的解码程序，将这份经过编码的东西还原，才能看到发信人要传送的信息是什么。也就是在发送时首先对邮件编码，接受时按指定的编码方式进行解码，得到邮件的真实内容。现在比较常用的标准是 RFC2045[16] 规定的 MIME（Multipurpose Internet Mail Extensions, 多目的网络邮件扩展）标准。它最主要的特点就是能够传输多媒体邮件，把多媒体文件作为一个附件与正文一起进行发送。

大量的 MIME 规范的重点在于多种内容类型的定义。表 3.1 列出了 MIME 规定的内容类型。主要有 7 种不同类型，15 种子类型。

表 3.1 MIME 内容的类型[16]

Table3.1 Type of MIME [16]

类型	子类型	说明
Text	Plain	未格式化的文本，可以是 ASCII 或 ISO8859
	Enriched	提供更多格式化的灵活性
Multipart	Mixed	不同的部分是独立的，但在一起发送。它们对接收方表示的顺序应当和在邮件消息中出现的一样。
	Parallel	与 Mixed 不同的是：Parallel 在对接收方交付部分时没有定义顺序。
	Alternative	不同的部分是相同信息的可替代版本。它们的顺序对于始发方可以增加可靠性，对于接收方的邮件系统应当宣示用户的最佳版本。
	Digest	与 Mixed 类似，但是每部分默认的类型/子类型是 Message/rfc822.
Message	Rfc822	正文本身是封装的消息，符合 rfc822。
	Partial	以对接收方透明的方式，用于允许大邮件项目的分段
	External-body	包含到在其他位置的对象指针
Image	Jpeg	JPEG 格式的图像
	Gif	GIF 格式的图像
Video	Mpeg	MPEG 格式的视频
Audio	Basic	单通道 8 位 ISDN mu-law 编码，采样速率为 8kHz
Application	PostScript	Adobe PostScript
	Octet-stream	由 8 位字节构成的常规二进制数据

MIME 消息通常包括文件附件和其他非 ASCII 数据。由于 RFC822[8]指定邮件消息中的所有字符必须是 ASCII 字符，所以必须使用编码算法将二进制数据转换为 ASCII 字符。MIME 定义了两类编码：Quoted-Printable 编码和 BASE64 编码。

MIME 标准现已成为 Internet 电子邮件的主流。它的好处是以物件作为包装方式，可将多种不同文件一起打包后传送。发信人只要将要传送的文件选好，它在传送时即时编码，收信人的软件收到也是即时解码还原，完全自动化，非常方便。当然先决条件是双方的软件都必须具有这种功能，要不然发信人很方便地把信送出去了，但收信人的软件如果没有这种功能，无法把它还原，看到的也就是一大堆乱码了。使用这种方式，用户根本不需要知道它是如何编码/解码的。即使只是用文字写的信，一样是打好包便寄出。如果是要寄多媒体文件，只要做选择文件的动作，选择完后寄出，其余的工作由电子邮件软件自动完成。

MIME 定义两种编码方式：Base64 与 QP(Quote－Printable)。QP 的规则是对

于资料中的7位无须重复编码，仅将8位的数据转成7位。QP编码适用于非ASCII码的文字内容，例如我们的中文文件。而Base64的编码规则，是将整个文件重新编码成7位，通常用于传送二进制文件。编码的方式不同会影响编码之后的文件大小。QP编码的文件比较小，而用Base64编码的方更有效。在这根据文件的扩展名来决定使用的编码方式。

Base64编码是把octets翻译成人们读不懂的字符,但它的编码和解码的算法都是很容易的，编码后只比原来的增大33%，但却提高了抗干扰性。

Base64的编码规则是：在改编码中，采用US-ASCII中的65个字符，可以用6比特组成用来表示的64个字符，第65个字符是"="，它被用来标出一个特别的处理过程。

在RFC1341[15]中介绍了Quoted-Printable编码规则。它的步骤如下：

①. 字符用 =XX 形式表示，其中 XX 是该字符的十六进制值，必须为 0-9 或者 A-F （使用大写字符）,除非有可替换说明，否则，此原则是强制性的。

②. 其中，十进制值 33-60 & 62-126(注意: 即不包含 '=') 可以作为标准ASCII 从而不进行转换。

③. 另外，十进制值 9-32 也可以作为制表和格式控制字符，从而不进行转换。(注意，这个不是必须执行的，即，也可以转换)

④. 由于在 RFC822[8] 协议中规定主体 body 文本中各行均有最大字符限制，因此，当主体文本中出现 CRLF 或者 LFCR 字符序列，或者单独的 CR 以及 LF 字符的时候，必须转换成对应的 "=0D=0A","=0A=0D","=0D","=0A" 等编码来表示。

⑤. (关于软回车的问题) Quoted-Printable 编码要求编码后每行最大字符数量不得超过 76 个字符。如果对大于该字符数量的行进行编码，则必须使用软回车。所以，对于某个以编码行的最后加上'='符号，则表示最后这个 '=' 是一个无意义的软回车。所以，如果一个尚未编码的行的内容如下的话:

Now's the time for all folk to come to the aid of their country.

那么在 Quoted-Printable 中可以表示为:

Now's the time =for all folk to come= to the aid of their country.

他提供了一种对过长的行进行编码并恢复到用户原来的输入内容的机制。虽然一行的末尾的 CRLF 不计入 76 个字符的限制之中，但是所有的其他字符，包括 '=' 符号都将被计算在内。

由于连字符号 '-' 在 Quoted-Printable 编码中表示他自己，所以当我们在对一个 multipart 实体的主体内容编码的时候，我们必须注意：我们决不能让一个 boundary 标志符出现在编码的主体部分！

在实现过程中定义了一些数据结构来存储这些信息。

邮件消息的结构体如下：

```
struct Message
{
  Mailbox *mailbox;        /* the mailbox this message belongs to */
  char *from;
  char *to;
  char *cc;
  char *subject;
  boolean mixed;
  char *message;
  char *attachment;
  char *filename;
  unsigned char state; // 3 = sent; 2 = updating; 1 = sending; 0 = no process;
};
```

MIME 的类型枚举体如下：

```
enum
{
  TYPEOTHER,
  TYPEAUDIO,
  TYPEAPPLICATION,
  TYPEIMAGE,
  TYPEMESSAGE,
  TYPEMODEL,
  TYPEMULTIPART,
  TYPETEXT,
  TYPEVIDEO
};
```

Encoding 类型枚举体如下：

```
enum
{
  ENCOTHER,
  ENC7BIT,
  ENC8BIT,
  ENCQUOTEDPRINTABLE,
```

```
    ENCBASE64,
    ENCBINARY
};
```

还有其他一些数据结构，如邮件状态标识，由于篇幅的原因就不在这详述了。在设计完相关的数据结构之后，每一封邮件的存储问题就解决了。之后邮件的发送接收问题就可以转变为，结构体的数据的读写问题了。发邮件就是往 Message 结构体中写数据，读邮件则相反。当然在读写过程中要对各个标识位进行判断，判断之后在进行相应的处理。

当然，写完数据结构之后，真正的数据存储并没有完成。这是因为要与服务器进行通信的数据必须是字符串。所以，要按照一定的格式把结构体中的数据转换为字符串类型的数据。

在这个转换过程中，其关键问题就是附件数据的存储。因为其他的各个数据项都很容易转换成字符串数据。上面已经提到，附件发送是利用 MIME 扩展实现的。其文件类型很多，为了能够统一的发送必须对其数据编码，已转化成统一的格式进行发送。

图 3.6，就是接收到的附件(.png 格式)显示的结果。

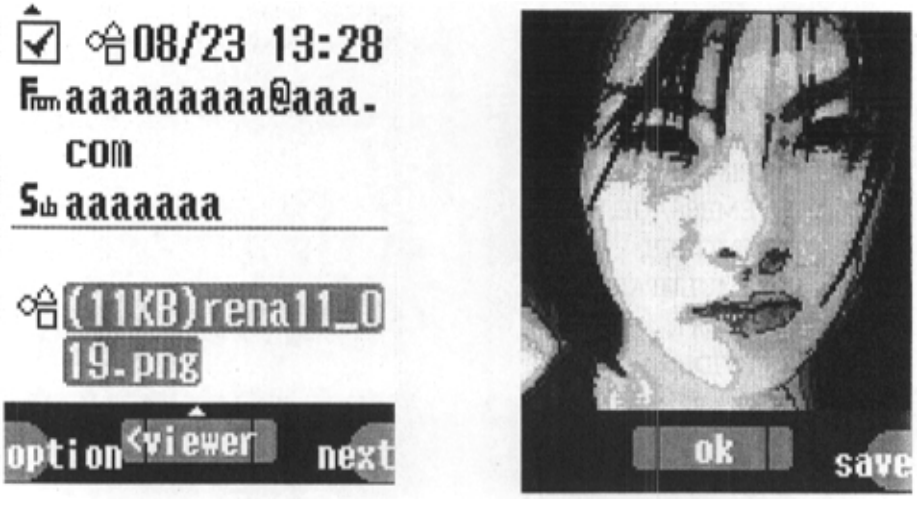

图 3.6 附件接收示意图

Fig.3.6 Result of Receiving Accessories

（3） 通信处理过程

从实现过程中可以靠到，上述协议的解析过程都是通过相应的命令来完成每

一次的通信过程。由于邮件处理的分成三层结构来处理，因此每一次通信都要分两次来完成，第一次是协议解析模块和协议通信模块进行通信，第二次是协议通信模块和服务器进行通信，他们通信的过程分别见图 3.7 和图 3.8。

其中：

虚线框中是解析模块，Net 是通信模块

MIRecv 表示从上层父 APP 接收消息

MISend 表示向上层父 APP 发送消息

NwRecv 表示从下层子 APP 接收消息

NwSend 表示向下层子 APP 发送消息

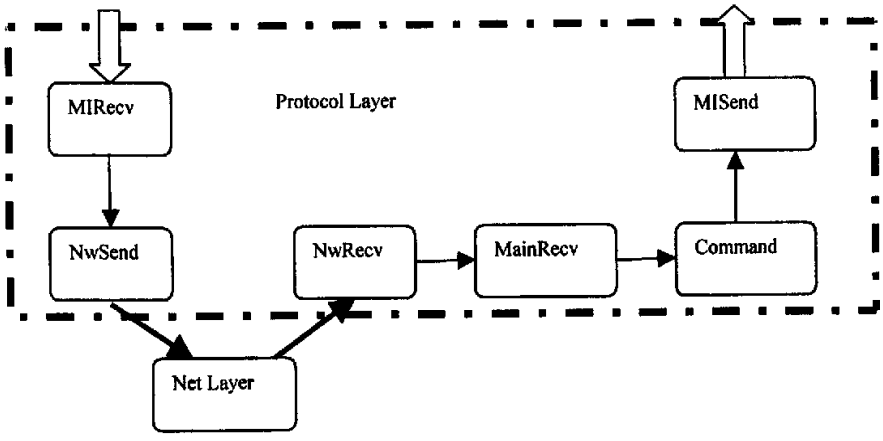

图 3.7 解析模块与通信模块通信过程

Fig3.7 Communicating Process between Parse Module and Communication Module

通信模块与服务器的通信是通过一个中间层完成的，这个中间层的作用非常的大，它是连接通信模块和服务器的桥梁，在这个中间层里首要判断网络状况，并根据当前的网络状态决定出服务器的当前状态和客户机的当前状态。依据这两个状态推测出下一个将要执行的命令是什么。并等到命令所需要的参数，设置好命令的格式。最后调用 Socket 来实现数据通信。

在各个状态判断的过程中，由于此系统是基于嵌入式系统的，所以必须对各个状态过程中的实时性很高的事件进行检测，如果有实时性比自己过的事件发生就必须中断当前的处理。并做一些备份性的工作。

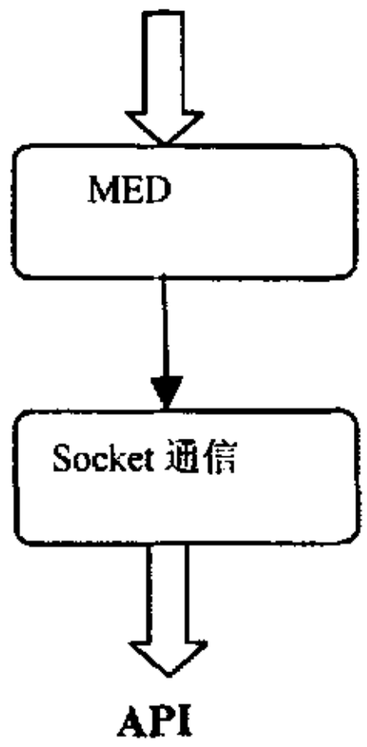

图 3.8 通信模块与服务器通信过程

Fig3.8 Communicating Process between Communication Module and Server

3.2 基于 IMAP4 协议的邮件接收功能的设计和实现

3.2.1 协议概述

IMAP4 由斯坦福大学开发的，是一种开放协议。作为 POP 协议的超集，IMAP4 允许用户远程创建，操纵，组织信箱，并实现邮件共享。IMAP4 有内建的数据库功能，可以记录查找，选取，删除邮件的规则。IMAP4 允许您存取邮件，但并不立即下载邮件，存取远程服务器上的邮件如同在本地一样. 这样一来, 无论您从公司的电脑，家里的电脑，笔记本电脑，还是从掌上电脑存取邮件，都不必自己复制邮件，因为服务器总是有该邮件的拷贝。而且，IMAP4 能识别 MIME，您可以只下载头部，然后选择想要的邮件。

IMAP 协议模型与 POP 协议模型非常的类似，具体模型见图 3.9。

由于现在支持 IMAP4 协议的邮件服务器还比较少，而且在此系统所使用到的邮件接收功能对服务器有特定的格式要求，所以在本系统中，作者自行编写了一个简化的基于 IMAP4 协议的邮件服务器，以方便对本系统的基于 IMAP4 邮件接收功能的测试。经过充分测试，结果表明，本系统已经完全可以利用 IMAP4 协议进行邮件的接收。

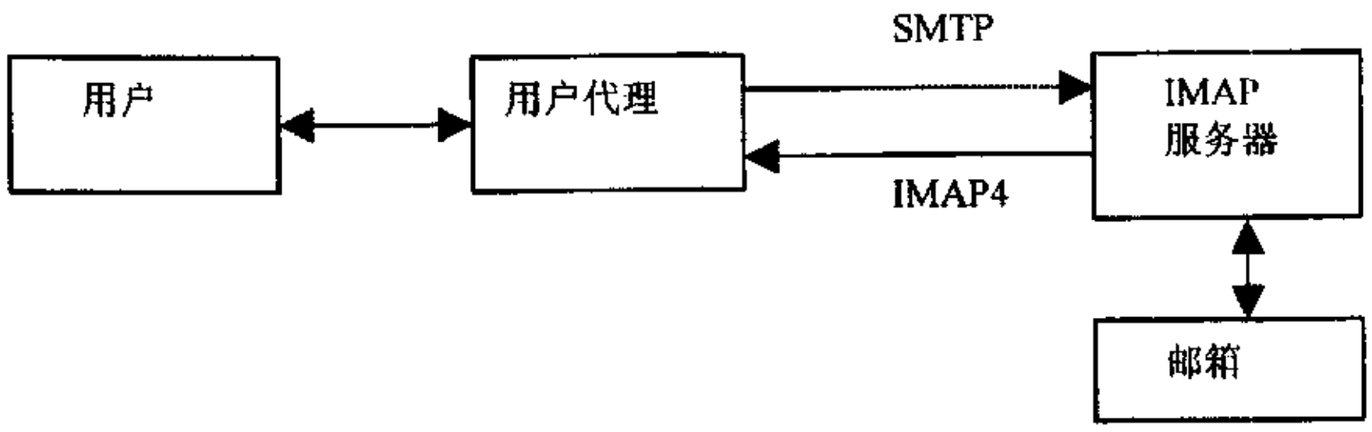

图 3.9　IMAP4 协议模型示意图

Fig.3.9 Model of IMAP4 Protocol

3.2.2 IMAP4 协议与 POP3 协议的区别

POP3 和 IMAP4 虽然都是邮件收信的协议，但是它们的能力和使用方法是差别很大的。

POP3 的功能集较少，容易实现。但是, POP3 也有很多缺点，一个问题是您除了用它收取邮件外，别的什么都不能做。您可以让 POP3 收取邮件后不删除它们。但是在服务器端就必须支持远程文件系统。如果邮件是 MIME (Multi-purpose Internet Mail Extensions -- 多功能因特网邮件扩展)类型的，您必须收取整个邮件。POP3 不支持邮件或信箱共享，而且它处理包含图片和多媒体附件的邮件的能力非常有限。 POP3 对只有一个信箱，并只从一个地方收取邮件的人非常适合。 但如果您有多个信箱，并经常在不同地方收取邮件，甚至使用不同的邮件客户程序，那么, IMAP4 将是您的选择。

POP3 只支持离线模式，而 IMAP4 支持在线和断线模式。在线模式需要与服务器保持网络连接，您可以不用下载邮件，但如果您要操作邮件，必须保持连接。断线模式 - 是离线方式和在线方式的组合，允许您下载邮件的全部或一部分，然后离线操作邮件。例如，您可以下载一个邮件，断线，读取邮件，然后把邮件放到另一个文件夹内。当您下一次连接时，客户端程序会通知服务器邮件位置的改变，服务器找到原先的邮件，然后把它移到相应的文件夹内。在这里，IMAP4 显示了它的同步能力，它能保证您在任何地方看到的邮件组织都是一样的。

防止邮件复制只是 IMAP4 的优点之一。在 IMAP4 中，您可以选择下载整个邮件，也可以只下载一部分。如果在收邮件时，时间紧急，这点是非常有用的。例如，当邮件包含很大的附件时，您可以先收取它的文本信息，而把附件留在服务器上。这样，您可以优先处理重要邮件。而收取附件的任务在后台执行。[28]

IMAP4 的另一个特性是支持邮件共享，当向一组人发送邮件时，不必给每一个人都保留一份拷贝。这是一个很好的群件解决方案 - 如果您正在管理新闻组，IMAP4 将是一个不错的选择。

总而言之, IMAP4 有下列优点：

· 支持在线，离线和断线模式存取邮件
· 更好的远程存取方式，您可以从多处访问邮件
· 支持邮件共享，及层次化的组织结构
· 有内建的数据库功能，可以定义规则
· 可以选择下载部分或整个邮件
· 不需要文件访问协议，也不需要知道服务器文件格式

当然，IMAP 的使用也有一些缺点：

· 从 POP3 转到 IMAP4 十分容易，但转回 POP3 十分困难
· 由于 IMAP4 需要服务器控制邮件和附件的存储，需要极多的服务器的资源

3.2.3 功能的设计和实现

3.2.3.1 功能的设计

在此模拟器上，基于 IMAP4 协议的邮件接收功能是通过三个步骤来实现的(见图 3.10)，发送接收模块是对发送接收过程的控制模块，它管理着整个邮件的发送接收过程，不管是从哪个地方启动邮件的功能，只要是这个功能要与服务器进行数据的交互，首先要进入的就是发送接收模块。它根据各个功能的不同调用不同协议的解析模块，控制协议解吸的启动，暂停和终止。当然还要对发送接收邮件的声音进行控制，并且还有各类事件的解析和异常的处理。当协议模块启动时，它首先转入协议解析模块对协议中的过程进行解析，这个过程就是根据 RFC 的标准来完成。协议解析的各个阶段都是利用一个或一些标准命令来实现，所以在这个阶段要处理协议各个阶段的命令，如果命令要与服务器进行数据的交互，就要在一定阶段发送通信信息以启动协议通信模块。交互完成之后，通信模块也要给解析 模块相应的回复。当数据在通信模块处理完毕后，协议分析模块分析处理完的数据，以确定是继续通信还是结束。协议通信模块的任务就是负责以服务器通信，并把服务器得到信息传回给协议解析模块。

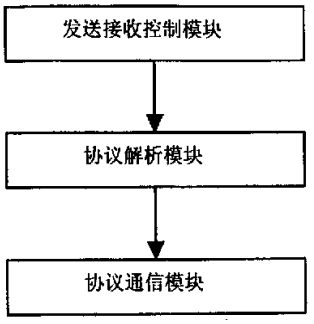

图 3.10 IMAP4 协议实现的步骤

Fig.3.10 Implementing Process of IMAP4 Protocol

IMAP 协议的解析也是在发送接收模块中调用的。它完成了以下的功能：

（1）. 接收新邮件（自动接收邮件）

(a) 使用[SACH]命令从服务器上取得未读的 UID(Unique ID)。

(b) 分别比较[SACH]命令指定的UID,收件箱内的邮件的UID及未收信列表中的 UID。如果都不存在此 UID，那就把这个 UID 加入到未收信别表中。

(c) 使用[FECH]命令取得在未收信列表中的 UID 指定的信件，如果服务器上没有此信件，则取未收信列表中下一个 UID 所指定的信件。如此继续处理，直到未收信列表中 UID 为空。附件的内容是否显示则参照 Email 设定中[Mail 收信方法]->[自动收信]的设定来决定。

（2）. 接收邮件本体

使用[FECH]命令取得在未收信列表中的 UID 指定的邮件本体，如果服务器上没有此信件，则取未收信列表中下一个 UID 所指定的信件。如此继续，直到未收信列表中 UID 为空。

如果邮件正文的大小超过 10K，那么就使用[STOR]命令，把服务器上的邮件取到客户机上，并在服务器上的此邮件标记为[已读]。

（3）. 接收邮件附件

使用[FECH]命令取得在未收信列表中的 UID 指定的邮件附件，在接收附件时，如果接收的文件名和文件夹中的文件名相同时，根据一定的命名规则变更文件名。

协议解析的过程可参照图 3.11，图 3.12，图 3.13，图 3.14，图 3.15。

(a) 接收新邮件（没有附件）过程

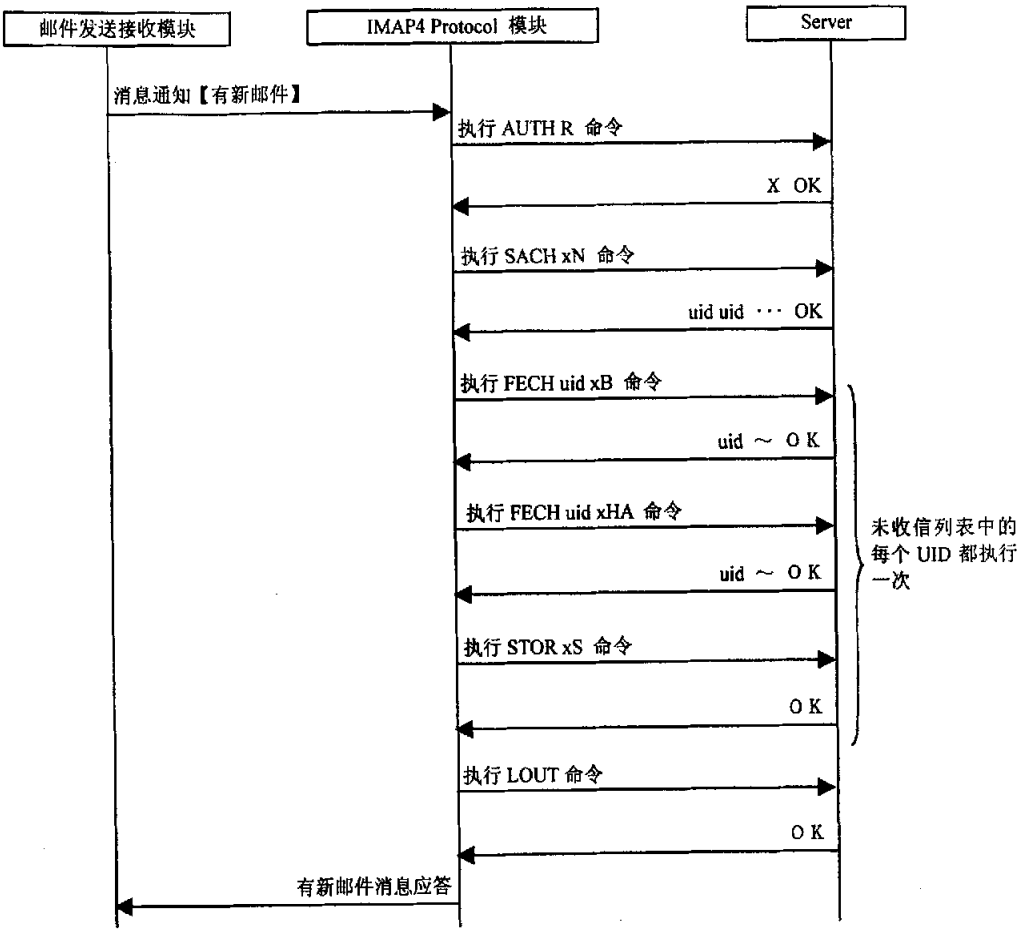

图 3.11 接收新邮件过程（没有附件）

Fig.3.11 Procedure of Receiving Email (Not Having Accessory)

(b) 接收新邮件（有附件）过程

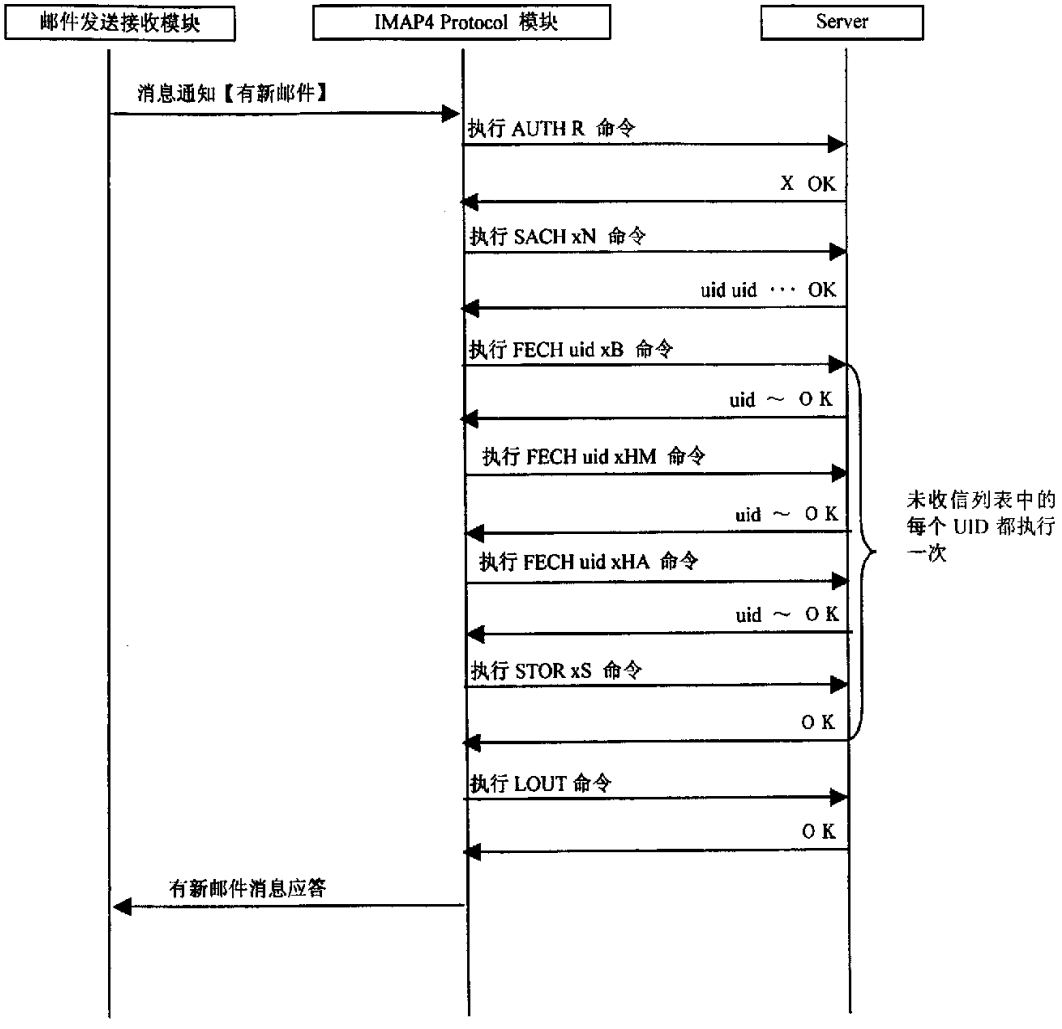

图 3.12 接收新邮件过程（有附件）

Fig.3.12 Procedure of Receiving Email (Having Accessory)

(c) 仅接收邮件正文（小于 10K）过程

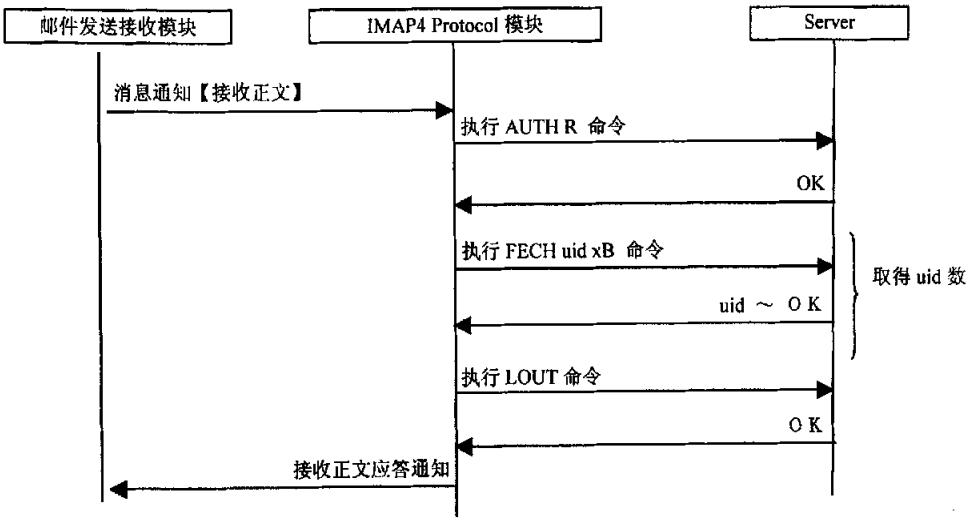

图 3.13 仅接收邮件正文过程(< 10K)

Fig.3.13 Procedure of Receiving Email Body Only (<10K)

(d) 仅接收邮件正文（大于 10K）过程

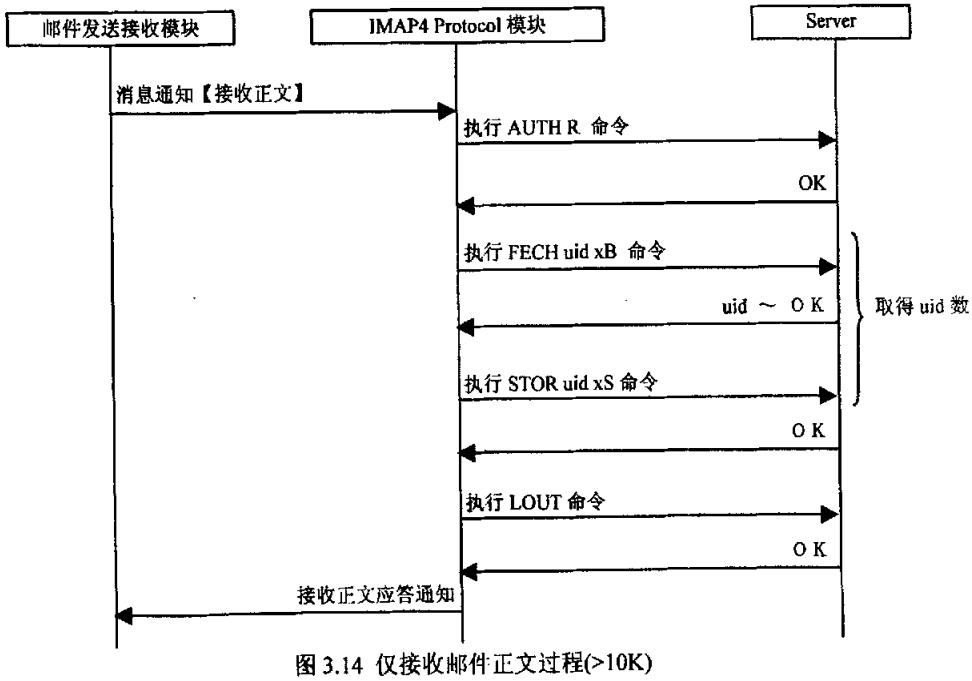

图 3.14 仅接收邮件正文过程(>10K)

Fig.3.14 Procedure of Receiving Email Body Only (>10K)

(e) 仅接收邮件附件过程

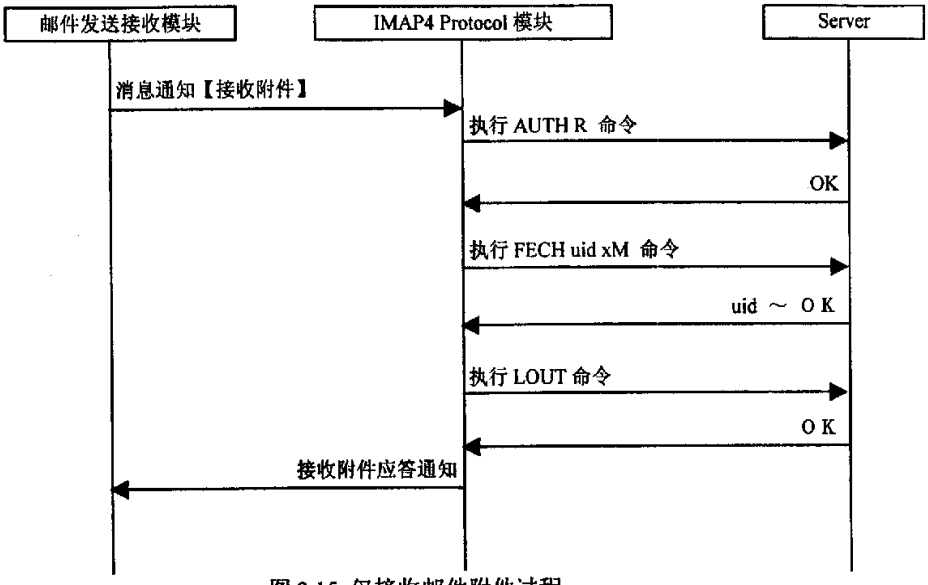

图 3.15 仅接收邮件附件过程

Fig.3.15 Procedure of Receiving Email Accessory Only

3.2.3.2 功能的实现过程

IMAP4 实现的过程基本上是按照 RFC2060[14]来实现的，下面是接收前部数据的过程，在实际应用中可能只需要接收文件头部，正文或附件。

```
S:  OK
C:  AUTH R
S:  1
S:  OK
C:  SACH xN
S:  1462
S:  OK
C:  FECH 1462 xB
S:  (1462{77}
("application" "OCT-STREAM" ("NAME" "pic.png") NIL NIL "base64" 10254 1)
)
S:  OK
```

```
C:  FECH 1462 xHA
S:  (1462{130}
Date: Thu, 22 May 2003 15:28:44
From: aaaaaaaa@aaa.com
To: aaaaaaaaa@aaa.com
Reply-To:
Subject: aaaaaaa
Cc:
BCc:
)
S:  OK
C:  FECH 1462 xHM
S:  (1462{65}
Content-Disposition: attachment;
    filename="rena11.png"
)
S:  OK
C:  FECH 1462 xB
S:  (1462{25}
This is a mail for test
)
S:  OK
C:  LOUT
S:  OK
```

从上面的通信过程可以看出，通信双方的信息量特别的少，这也是因为嵌入式系统的资源有限决定的。

3.2.3.3 实现过程中的关键问题及其解决方案

（1）系统实时性控制

嵌入式系统对实时性的要求都比较的高，所以在设计和实现嵌入式系统软件时，就不得不考虑系统的实时性问题。在本系统的 TCP/IP 应用层协议的各个模块中，都有针对实时性事件处理模块。

在本系统的执行过程中，优先级比较高的事件主要有来电、没电、圈外和按键。它们要求实时性非常的高，一般的功能模块在遇到以上的四种情形时，都要进行相应的中断处理。本模块当然也不例外。为了说明方便，就以 Email 发送的过程中对各种中断的处理来进行说明，其他的过程（如 Email 接收，Chat Mail 发送）

与之非常的相似就不重复介绍了。

① 来电处理

当一个客户机正在发送邮件的时候，接收到一个电话。由于来电的处理的优先级很高，必须立即处理，不管现在发送邮件进行到那个阶段，客户机的应用程序管理模块都会立即中断邮件发送模块，切换到来电状态。此时在没有发送完成的邮件被存储在发件箱，并表示此邮件为未发送完成状态，下次可以重发。而在服务器那端，在一定的时间段内一直监听发送邮件的端口，如果在这段时间内没有收到任何来自客户机的命令，它将重新进行复位，等待客户机下一次的发送请求。

② 没电处理

当用户在发送邮件的时候，发现电量不足，这时模拟器就会从外部得到一个设备事件，指明现在的客户机设备有故障，不能继续发送邮件。这时就会在 LCD 上显示一个警告提示框，告诉用户现在设备将在一段时间之后自动关机。此时应用程序管理模块就会及时地中断邮件发送模块，当然也不论现在发送模块所处的阶段。没有发送完成的邮件被存储在发件箱，并表示此邮件为未发送完成状态，下次可以重发。而在服务器那端，在一定的时间段内一直监听发送邮件的端口，如果在这段时间内没有收到任何来自客户机的命令，它将重新进行复位，等待客户机下一次的发送请求。

③ 圈外处理

当用户在发送邮件的时候，发现现在已经处于服务器的范围之外，这时模拟器就会从外部得到一个设备事件，说明现在不能继续发送邮件。这时也会在 LCD 上显示一个警告提示框，告诉用户现在已经超出服务器服务的范围，请下次再发送邮件。此时应用程序管理模块就会及时地中断邮件发送模块，当然也不论现在发送模块所处的阶段。没有发送完成的邮件被存储在发件箱，并表示此邮件为发送失败状态。而在服务器那端，在一定的时间段内一直监听发送邮件的端口，如果在这段时间内没有收到任何来自客户机的命令，它将重新进行复位，等待客户机下一次的发送请求。

④ 按键处理

在用户发送邮件的过程中，用户可能会发送地址错误，或者是发送的内容错误等等其他的一些原因致使用户想取消或终止当前的发送。用户可以按 Clear 键取消当前的发送，或者按 Power 键强制终止当前的发送。在此时也只相应这两个按键事件，其他的按键事件将会被存储，等到邮件发送完成之后处理。

当用户按下 Clear 键来取消当前操作时，此时不用调用应用程序管理模块，而直接在发送模块中判断，如果是按键是 Clear 键，则把当前的发送的状态设置为

Cancel 状态，并向服务器发送退出命令。此时程序将返回到发送模块，用户可以进行修改邮件，并再次发送。

当用户按下 Power 键来终止当前的发送过程，那么直接在发送模块就把当前的发送的状态设置为强制终止状态，并向服务器发送退出命令。此时程序将返回到主菜单模块，并把这封邮件设置为未发送邮件。

在按键事件发生时，服务器不用作任何特殊的处理。

（2）多个协议并存控制

前面已经提到，本系统中同时支持两个接收邮件的协议——POP3 和 IMAP4。所以在应用的过程中必须加于判断。在此就设计了一个枚举型数据：.

```
enum{
    EMAIL_SETTING_PROTOCOL_IMAP4      =  0,    /* IMAP4 协议用 */
    EMAIL_SETTING_PROTOCOL_POP3                /* POP3 协议用*/
};
```

并在 DB 中的协议设定结构体中增加一个 unsigned int 型变量 ProtocolSetting 来存储这个数据。为了体现程序的封装效果，为 DB 之外的任何模块提供统一的接口，编写了两个 API 函数 ApiEmailLclSettingGetProtocolSetting（BYTE setting）和 ApiEmailLclSettingSetProtocolSetting（void），以供外部模块调用。其中函数 ApiEmailLclSettingGetProtocolSetting（BYTE setting）只在 Email 设定模块修改接收邮件协议时调用，而调用函数 ApiEmailLclSettingSetProtocolSetting（void）的地方比较多，主要的情形有：

① Email 模块初始化的时候就要判断现在系统支持的接收协议是什么。

② 在修改接收邮件接收协议设定的时候也要判断协议是不是改变了，如果没有改变就不用重新设定了。

③ 在两个接收邮件协议处理存在差异的时候更需要调用这个函数来判断了。

④ 在邮件接收模块要启动子模块来解析邮件之前也必须先判断当前的接收协议。

⑤ 由于两个协议执行的中断处理也不尽相同，所以当系统在执行中断时，也要根据不同的协议设定来执行不同的处理。

同时，在本系统中，发送和接收是在同一模块中处理的，所以，在适当的是欧还必须区别发送和接收的协议。此时是通过函数 EmailSendRecvGetType()是实现此功能，如果此函数返回的值是 EMAIL_SENDRECV_RECV 则表示现在正在接收邮件，否则就是在发送邮件了。图 3.16 就是本系统多个协议共存时判断当前的协议的数据流程图。

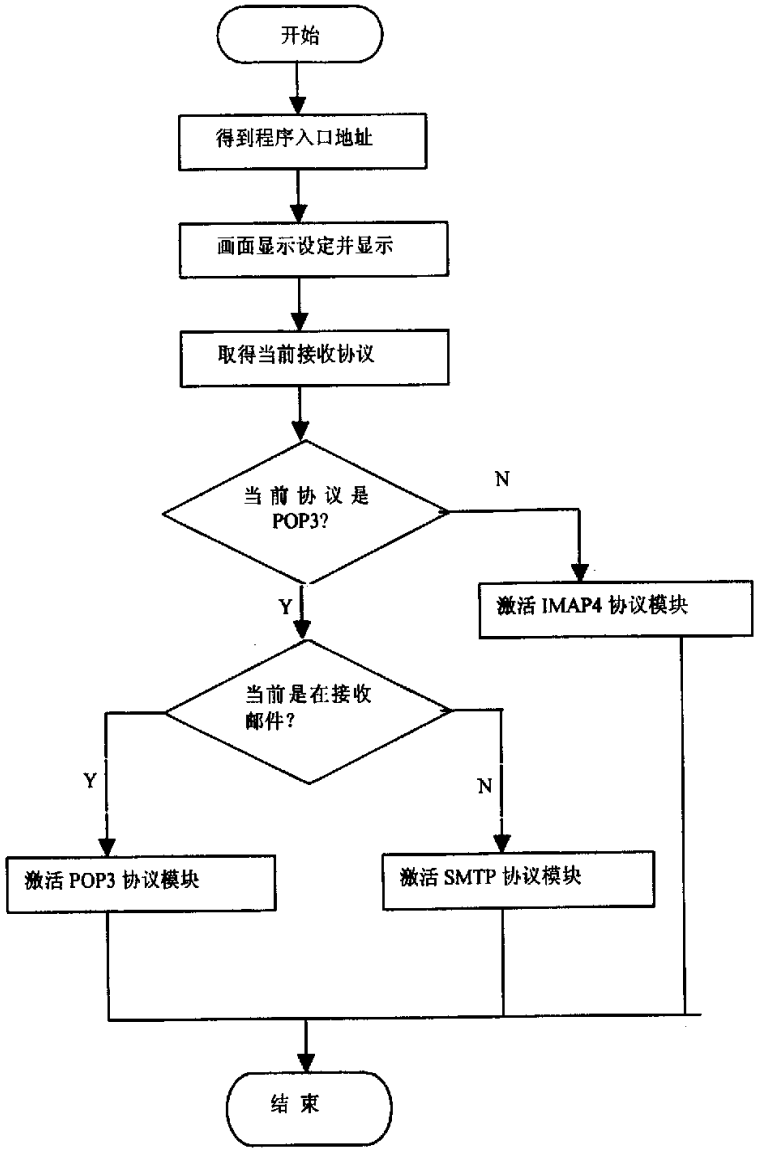

图 3.16　当前使用协议判断数据流程图

Fig3.16 Data Flow Diagram of Getting Current Protocol

（4）IMAP4 邮件服务器的设计和实现

本论文设计并实现了一个简化的 IMAP4 邮件服务器（见图 3.17），经过测试表明，它完全能够实现邮件正文接收，邮件附件接收，邮件自动接收，邮件部分接收，邮件回复，邮件转发等等 IMAP4 协议和其他邮件协议所规定的大部分的功能。其工作的流程图可参照图 3.20。

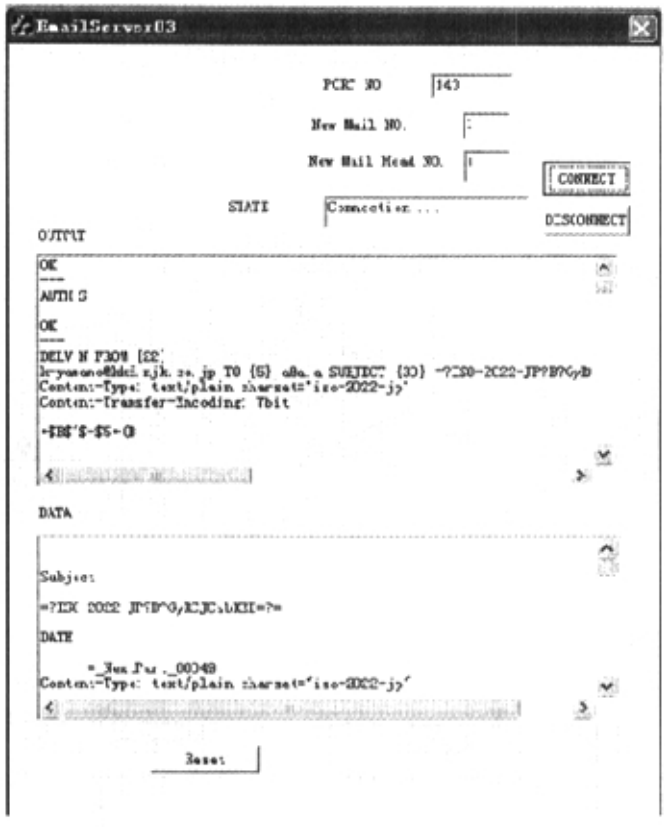

图 3.17 IMAP4 邮件服务器

Fig.3.17 IMAP4 Email Server

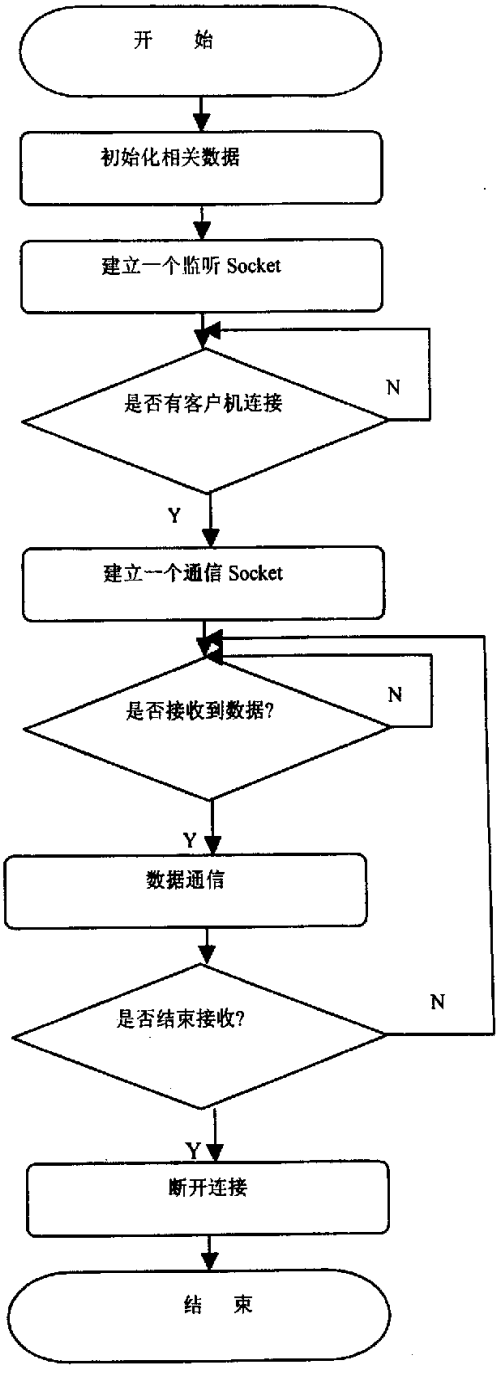

图 3.20 服务器工作流程图

Fig.3.20 Work Flow Diagram of Server

第四章 嵌入式系统模拟器下聊天功能的设计和实现

4.1 协议概述

IRC(Internet Relay Chat, 网络中继聊天)[18]协议用于文本交谈被设计出来已经有许多年了。

IRC 协议是基于客户服务器模型的，可以很好地分布式地在许多机器上运行。一个典型的设置涉及一个进程(服务器)，它作为中心点接受客户(或其它服务器)的连接，并且实现要求的消息传送/多元技术和其它的功能。这种分布模型，由于它要求每个服务器都拥有全局状态信息，限制了一个网络所能达到的最大规模，因此是此协议最令人不能容忍的问题。现存的网络能够以难以置信的速度持续增长，我们必须感谢硬件制造商们给了我们比以往更加强大的系统。

IRC 协议本身就是一个电话会议系统（虽然使用的是 C/S 模式），所以非常适合分布式的多个机器上使用。一个典型的建立包括一个单一的处理器（服务器）形成一个中心节点，客户机（或其他服务器）连接这个节点去获取消息或发送/群发等功能。

IRC 协议已经在使用 TCP/IP 网络协议的系统中应用了。虽然 TCP/IP 协议并不是必须的，但是现在 IRC 操作仅使用在这个方面。

4.2 功能的设计和实现

4.2.1 功能的设计

在本系统中，IRC 并用于实现 Chat Mail 客户端与 Chat Mail 服务器之间直接发送命令和应答的应用程序。

IRC 模块由以下的 5 个功能构成。

①单发 Mail 发送功能

②连发发信功能

③连发收信功能

④组送信功能

⑤存储发信功能

IRC 模块是后台运行的程序，他的生命周期是从与服务器的 PPP[10]链接完成到链接释放这个期间。但是 IRC 的各个功能以及各个功能的每个执行操作都不可

以同时进行。换句话说，就是在某一个时刻只有一个功能的一个操作在进行。

下面将说明 IRC 模块的各个功能。但是各个功能的事件，命令，各个功能的处理过程请参照 IRC 顺序图。

(1) 单发 Mail 送信功能

先选定一个要接收 Mail 的设备，然后执行单发送信的功能，根据 Chat Mail 模块处理就启动了单发 Mail 送信功能。IRC 功能模块就是连接到 Chat 服务器，并向服务器单发送信。送信完毕后，根据从 Chat 服务器接收到的应答信息来启动相应的子 APP。如果送信没有完成，就发送一个送信失败的应答。

(2) 连发发信功能

从此嵌入式设备到其指定的要与其聊天的另一个对应的嵌入式设备发送连发 Mail 后，根据 Chat Mail 处理模块就启动了连发送信功能。IRC 模块于是就连接到 Chat 服务器，并向服务器发送 First Mail。发送完 First Mail 之后，对应的嵌入式设备就与服务器连接，这样就可以进行连发送信和收信的处理了。

(3) 连发收信功能

从对应的嵌入式设备接收到 First Mail 并发送应答 Mail 之后，并根据 Chat Mail 模块的处理启动了连发接收功能。然后 IRC 模块就与服务器进行连接，连接完成之后就可以进行连发发送和接收处理了。

(4) 组送信功能

组送信要先在设定对应组中的对应的设备名（每个组最多 10 个设备名）。然后根据提起出组中的各个需要通信的设备名，进行单发送信。这样就在 Chat Mail 处理模块过程中启动了组送信功能。IRC 模块于是就连接到 Chat 服务器，然后就发送组送信 Mail。送信完毕后，根据从 Chat 服务器接收到的应答信息来启动相应的子功能模块。如果送信没有完成，就发送一个送信失败的应答。

(5) 存储发信功能

单发送信失败或连发送信失败后，所发送的信息被存储，下次可以再次发送，在这次发送时，Chat Mail 启动的就是存储功能处理。然后 IRC 模块就与服务器进行连接，进行存储 Mail 的发送。送信完毕后，根据从 Chat 服务器接收到的应答信息来启动相应的子功能模块。如果送信没有完成，就发送一个送信失败的应答。

协议解析的顺序图可以参照图 4.1，图 4.2，图 4.3，图 4.4，图 4.5。

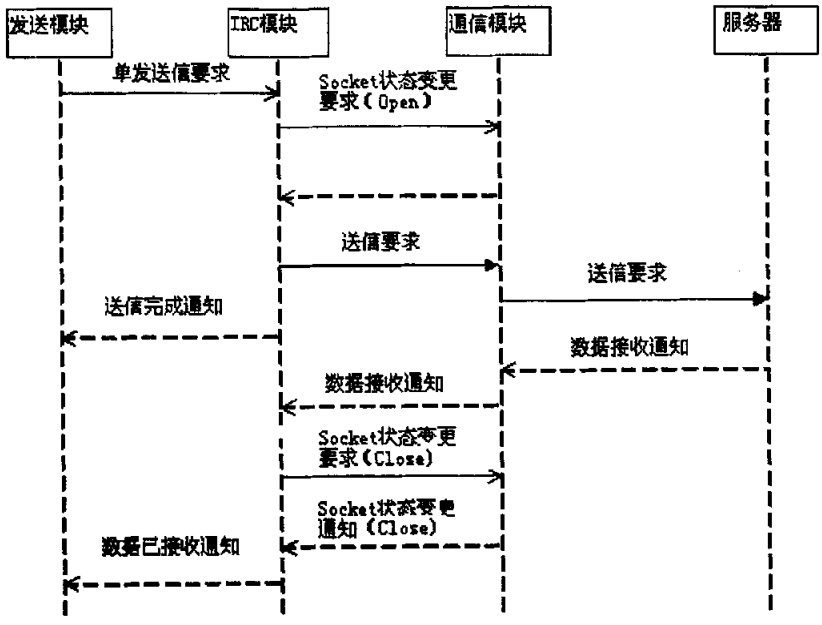

Fig.4.1 Sequence Diagram of Single Mail Sending

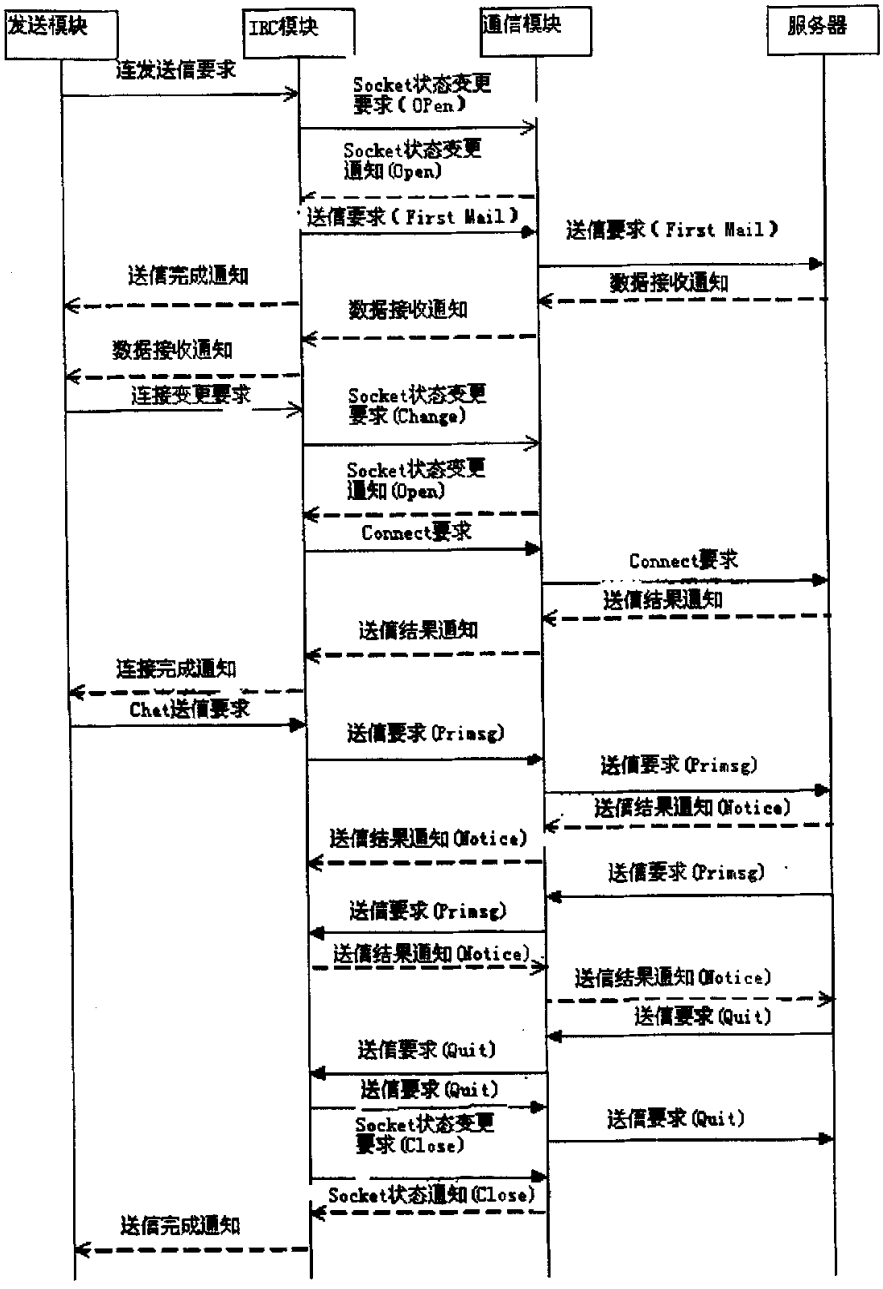

图 4.2 连发送信顺序图

Fig.4.2 Sequence Diagram of Chat Mail Sending

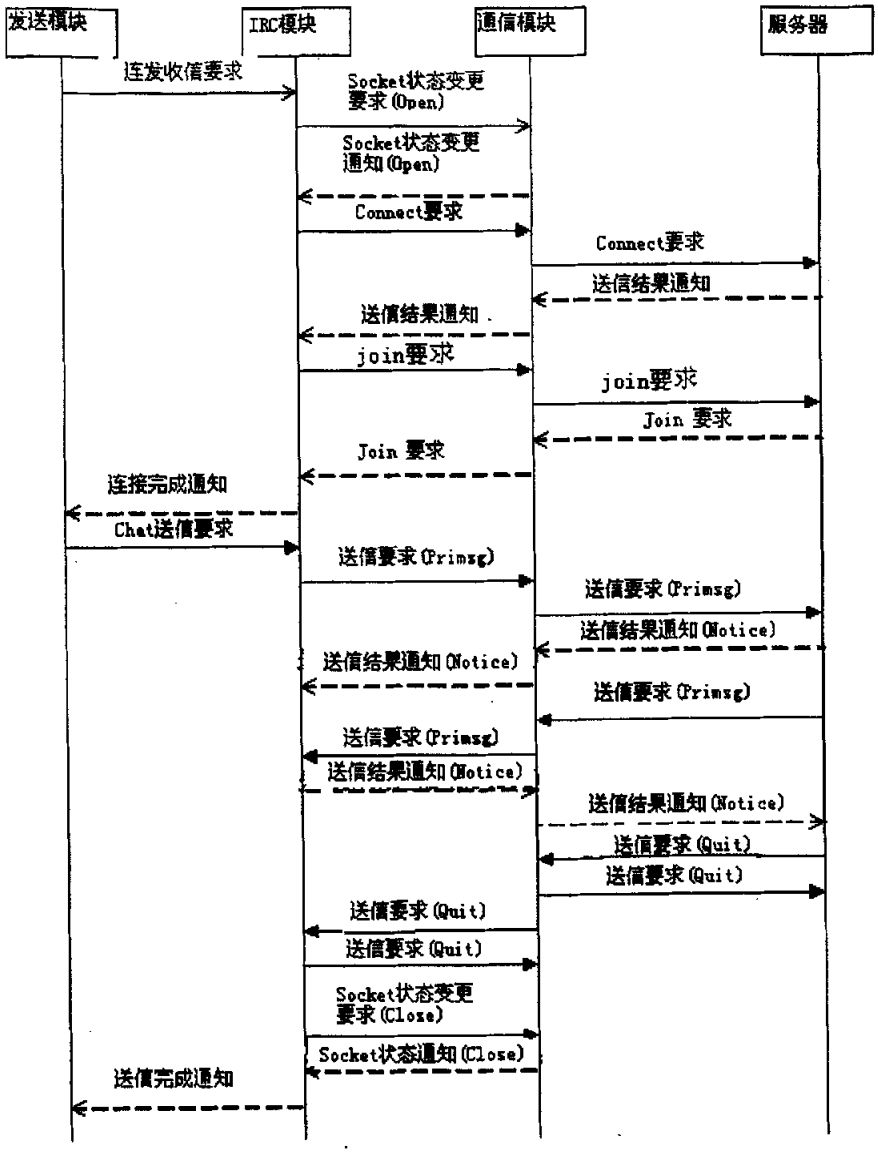

图 4.3 连发收信顺序图

Fig.4.3 Sequence Diagram of Chat Mail Receiving

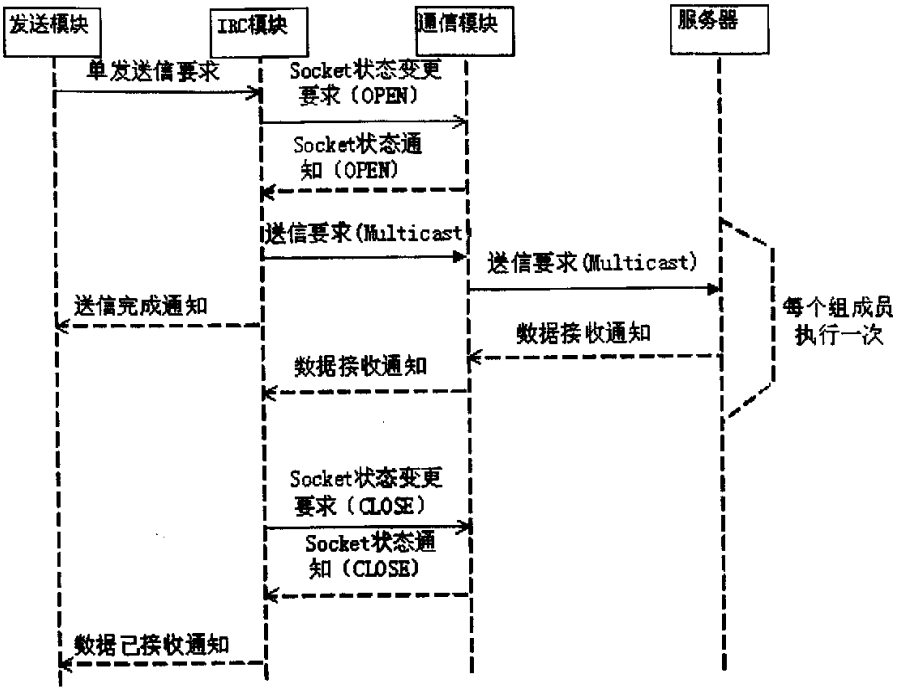

图 4.4 组送信顺序图

Fig.4.4 Sequence Diagram of Group Sending

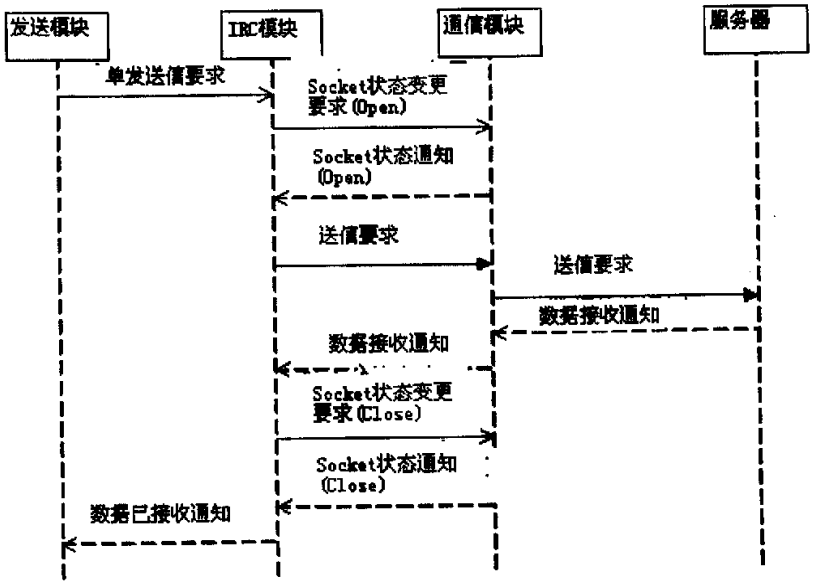

图 4.5 存储送信顺序图

Fig. 4.5Sequence Diagram of Store Sending

4.2.2 功能的实现过程

依照上节所设计的结构和过程，可以很方便的实现出聊天功能。下面就是一个聊天过程的执行情况。

聊天的现实过程为：

A : Hello

B : Hello

A(B): ……

A: Bye

B :Bye

其中 A，B 是虚拟的聊天者。

在实现过程中，这个聊天过程就要借助一个服务器(SV)来实现，聊天的发送端（MS）要先跟这个服务器通信，然后由这个服务器与接收方进行通信，最后完成聊天过程。下面就是整个的处理：

[MS to SV]: squery 01234567 1234567 :(186)Hello
[SV to MS]: snotice 192.168.31.65 Disp
[MS to SV]: NICK B00039447
USER I 0 * :I
[SV to MS]: zjg 001 B00039447 :Welcome to the Internet Relay Network B00039447! ~I@zjg
[MS to SV]: JOIN #012345671234567
[SV to MS]: B00039447!I@zjg JOIN #012345671234567
:zjg 353 B00039447=#012345671234567 : B00039447 M00008707
:zjg 366 B00039447 #012345671234567 :End of NAMES list.
[SV to MS]: M00008707!I@zjg PRIVMSG #012345671234567 :(184)Hello
[MS to SV]: NOTICE #012345671234567 :ACK
[MS to SV]: PRIVMSG #012345671234567 :(186).....
[SV to MS]: M00008707!I@zjg NOTICE #012345671234567 :ACK
[SV to MS]: M00008707!I@zjg PRIVMSG #012345671234567 :(184)Bye
[MS to SV]: NOTICE #012345671234567 :ACK
[MS to SV]: PRIVMSG #012345671234567 :(186)Bye
[SV to MS]: M00008707!I@zjg NOTICE #012345671234567 :ACK

4.2.3 实现过程中的关键问题及其解决方案

（1）服务器的设计

在实现的过程中，为了测试协议的功能，自行设计了一个 Chat Mail 服务器与

之通信。见图 4.6。

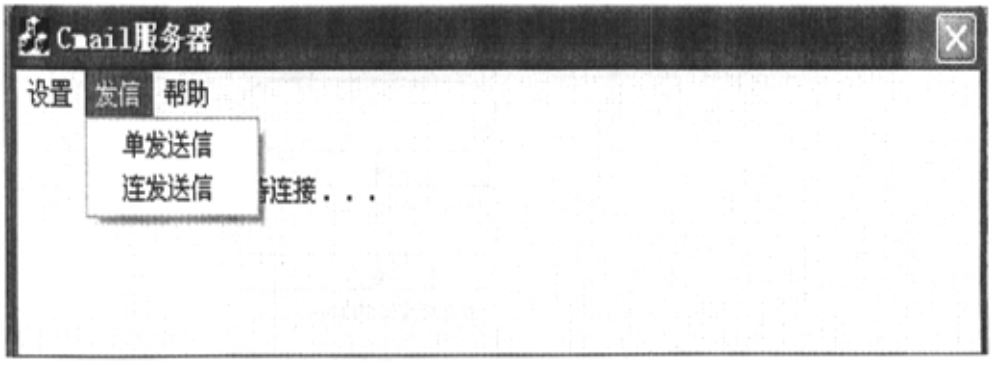

图 4.6 Chat Mail 服务器

Fig.4.6 Chat Mail Server

由于 这个服务器必须对上述五个功能进行相应处理，因此在处理的过程中必须对各种通信功能进行判断。在此处是对每个接收到消息的头部进行分析。

如果前几个字母是" smtran"，则表示是 Single Message（单发送信）；如果是" squery" 表示连发送信。组通信和存储通信都可以看作是单发送信的变体。

服务器在处理单发送信（包括，组通信和存储通信）是只要在向客户端发送一个回复，表示当前服务器可用就可以完成任务，而在连发送信过程中，必须有相应的聊天室进行连续的通信。如果是发送端，就必须建立一个聊天室（其实就是另外建立一个端口处理数据），而在接收端，就必须加入这个聊天室，这个聊天室的名字是用发送者和接收者的 ID 结合而成。

“#03346511101234567”就是一个聊天室的名字。

#：是一个标志位，说明是一个聊天室名；

03346511：是发送者 ID；

01234567：是接收者 ID；

则 JOIN #03346511101234567 就表示要加入这个聊天室了。

（2）模拟器的被动启动

IRC 协议是一个聊天协议，这就意味着客户端和服务器端的通信是双向的。这就与 Email 实现的协议有所不同，在 Email 实现过程中，总是客户端发送一个命令，服务器根据这个命令给一个应答。而在基于 IRC 协议聊天的过程，服务器端还需要模拟一个聊天客户端，并向客户端发送信息。这就意味着存在这样一种情况，一个客户端本来处于待机状态，而现在有连接要求，这样从服务器上就要发送一个消息以启动这个客户端程序，使之处于聊天状态。由于模拟器都有一个固定的

窗口句柄，所以可以使用窗口句柄来模拟这个功能。其启动客户端程序的过程如图 4.7。

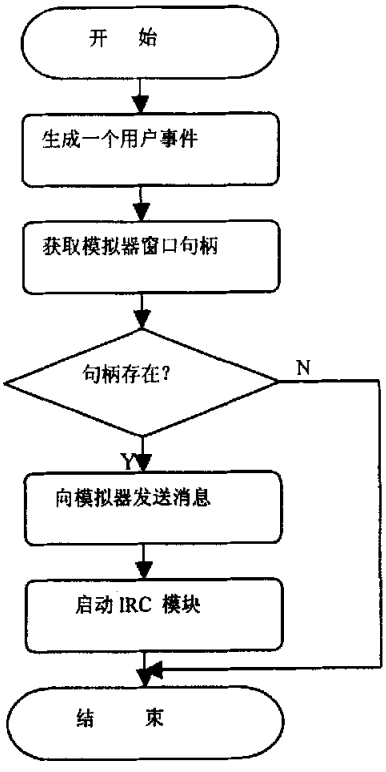

图 4.7 启动客户端程序过程

Fig.4.7 Process of Starting Client

（3）事件及相关状态控制

在实现的过程中，一个非常复杂的问题就是各个事件的区分及其各个状态的划分。一共定义了五个事件，18 个状态：

```
enum{
        IRC_EVENT_1,                    // 单发送信
        IRC_EVENT_2,                    // 连发送信
        IRC_EVENT_3,                    // 连发收信
        IRC_EVENT_4,                    // 组送信
        IRC_EVENT_5,                    // 存储发信
        };
enum{
```

```
    IRC_STATUS_1,                    // 初始状态
    IRC_STATUS_2,                    // 等待 Socket Open 通知（客户端）
    IRC_STATUS_3,                    // 信息发送中
    IRC_STATUS_4,                    // 等待连接服务器
    IRC_STATUS_5,                    // 等待 Socket Open 通知（服务器）
    IRC_STATUS_6,                    // 服务器连接中
    IRC_STATUS_7,                    // chat 进行中
    IRC_STATUS_8,                    // 未使用
    IRC_STATUS_9,                    // 未使用
    IRC_STATUS_10,                   // 未使用
    IRC_STATUS_11,                   // 未使用
    IRC_STATUS_12,                   // 未使用
    IRC_STATUS_13,                   // 未使用
    IRC_STATUS_14,                   // 未使用
    IRC_STATUS_15,                   // 未使用
    IRC_STATUS_16,                   // 未使用
    IRC_STATUS_WAIT_SOCKET_CLOSE,    // 等待 Socket Close 通知
    IRC_STATUS_END,                  // 结束状态
};
```

当然上述的各个状态并不是在每个事件中都存在，例如在 IRC_EVENT_1（单发送信）中仅存在 IRC_STATUS_1，IRC_STATUS_2，IRC_STATUS_3，IRC_STATUS_WAIT_SOCKET_CLOSE 这几个状态。从上面我们也可以看到，有 9 个状态暂时还没有被使用，他们预留到以后软件升级上使用。

第五章 嵌入式 Linux 电子邮件功能的设计和实现

在 Linux 下的实现包括收发邮件界面，邮件编辑界面，选项界面等几个部分。

(1) 收发邮件界面用于管理邮件，类似于 OUTLOOK 的做法。

(2) 邮件编辑界面用于书写电子邮件。

(3) 选项界面用于设置收发电子邮件时用到的参数，例如 SMTP 服务器设置，POP3 服务器设置。

5.1 功能的设计

5.1.1 界面设计

软件是人机交互的手段，在很大程度上，软件体现一个系统的可操作性。作为用户来说，和系统的交互是通过软件完成的。结合目前的实际，我们认为软件的界面很重要，可以说在某种意义上决定了用户对产品的好恶。因此我们尽量使界面直观，功能一目了然，方便用户的操作。

5.1.2 模块化设计

为了使软件代码具有良好的可读性，可维护性，特别是为了以后的软件升级，就必须采用模块化设计。而且，当几个人共同完成一个软件工程的时候，只有采用模块化设计，才能保证整个软件内部的良好结合。采用从顶层划分，然后再进行各个子模块的设计。虽然，这样不能保证整个软件的代码是最简单的，但是减少了调试的难度，也方便了以后的维护。在模块化设计时必须遵守以下几个基本原则：

（1） 函数功能单一。这种方式保证灵活性，可执行性和易于使用性，而且也保证了每个函数功能都是最小划分。

（2） 函数之间要相互独立。如果两个函数或多个函数要联合起来使用，则要仔细划分每个函数的功能。

软件的功能模块图见图 5.1。整个软件由多个模块共同配合完成，包括邮箱管理，发送邮件，接收邮件等。其实现过程与前面介绍的协议解析过程非常类似，在这就不重复讲述了。

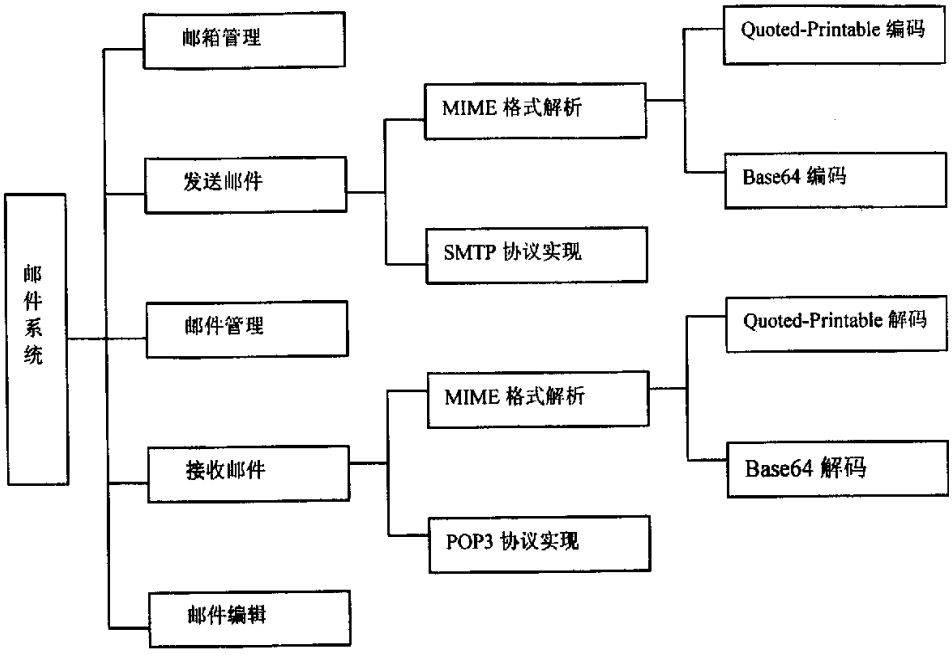

图 5.1 邮件系统功能模块图

Fig.5.1 Mail System Function Module Diagram

5.2 功能的实现过程

5.2.1 界面的实现

移植到 Linux 上之后，软件的界面必须改变，本论文的界面都是使用 GTK+[23] 实现的。图 5.2，图 5.3，图 5.4 是相关界面的剪辑。

图 5.2 收发邮件界面

Fig.5.2 Sending Mail Interface

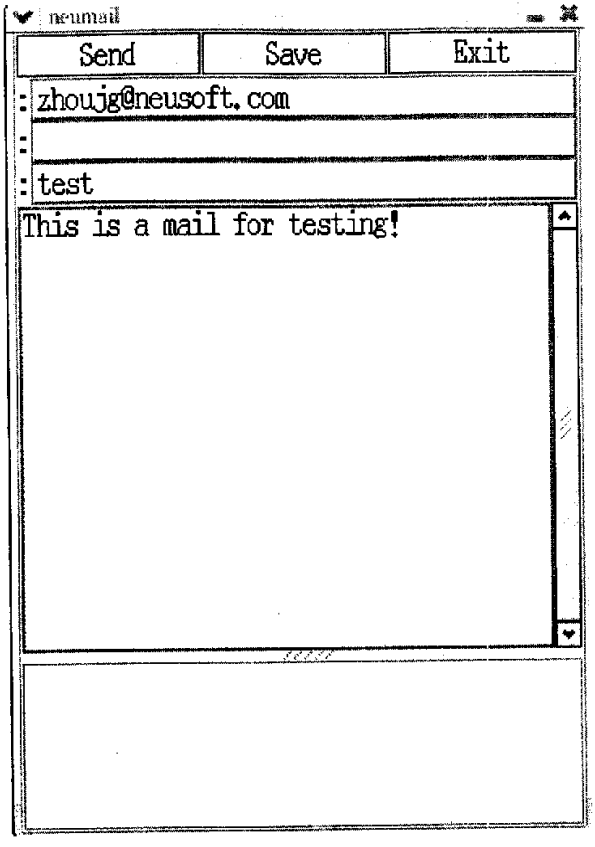

图 5.3 邮件编辑界面

Fig.5.3 Editing Mail Interface

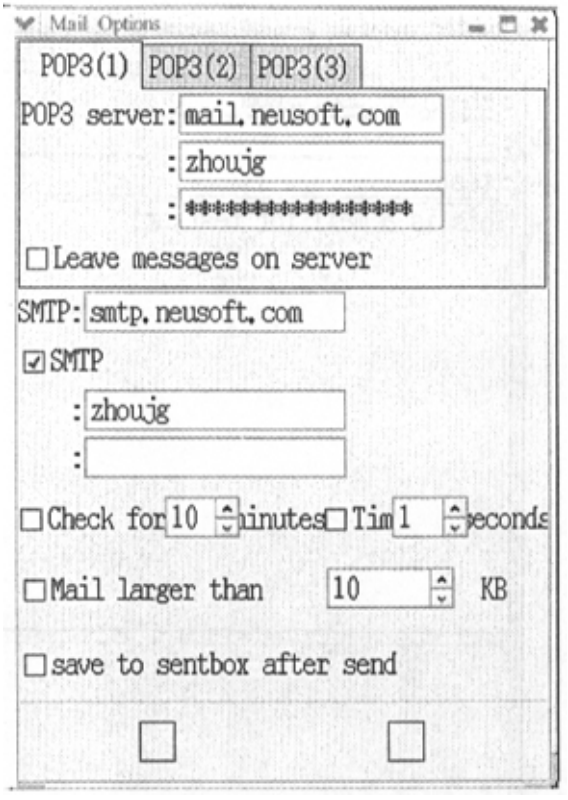

图 5.4 选项界面

Fig.5.4 Options Interface

5.2.2 Socket 层的重写

在模拟上实现的网络通信是通过调用模拟器上专用的 Socket API 实现的，如果移植到 Linux 上，这些 API 都不能直接调用，必须重新设计。

涉及到的主要函数有：

（1）创建一个 Socket

```
int open_socket(char *host, int port);
```

（2）向 Socket 写数据

```
int sock_write(int socket, char *buf, int len);
```

（3）从 Socket 中读出一行数据

```
int sock_readline(int socket, char *buf, int len);
```

（4）从 Socket 中读出所有数据

```
int sock_readall(int socket, char *buf, int len);
```

5.3 实现过程中的关键问题及其解决方案

（1） GTK+中的事件处理[23]

编写 GUI 应用程序的主要任务包括两点：创建窗口部件；编写代码以处理 callback 函数中的事件。GTK 使用“信号处理函数”来处理程序中的事件。GTK 的 callback 函数是在头文件 gtkwidget.h 中定义，如下所示：

```
typedef void (*GtkCallback) (GtkWidget *widget, gpoint data);
```

GTK 函数 gtk_main 使用 gtk_signal_connect 函数将事件和 GTK 窗口部件绑定，通过这种方式处理所有的注册事件。函数原型如下：

```
guint gtk_singal_connect(GtkObject *object,
                         const gchar *name,
                         GtkSingalFunc func,
                         gpointer func_data);
```

第一个参数是指向某个 GtkObject 的指针，然而在实际中传递的是 GtkWidget 对象的地址。GtkWidget 在其结构定义的起始处包含了一个 GtkObject，因此将一个 GtkWidget 指针强制转换成 GtkObject 指针时是安全的，第二个参数是事件类型的名称。第三个参数是处理函数。第四个参数是处理函数的参数。

以下本文所注册的事件：

```
gtk_signal_connect (GTK_OBJECT(list),
                    "select_row",
                    GTK_SIGNAL_FUNC(on_row_selected),
                    NULL);
gtk_signal_connect (GTK_OBJECT (list),
                    "click_column",
                    GTK_SIGNAL_FUNC (on_click_column),
                    NULL);
gtk_signal_connect(GTK_OBJECT(window),
                    "delete_event",
                    GTK_SIGNAL_FUNC (email_quit),
                    NULL);
gtk_signal_connect(GTK_OBJECT(window),
                    "configure-event",
                    GTK_SIGNAL_FUNC (configure_cb),
                    NULL);
gtk_signal_connect(GTK_OBJECT(inbox_button),
```

```
                    "clicked",
                    GTK_SIGNAL_FUNC (inbox_button_cb),
                    NULL);
gtk_signal_connect(GTK_OBJECT(draftbox_button),
                    "clicked",
                    GTK_SIGNAL_FUNC (draftbox_button_cb),
                    NULL);
gtk_signal_connect(GTK_OBJECT(sentbox_button),
                    "clicked",
                    GTK_SIGNAL_FUNC (sentbox_button_cb),
                    NULL);
gtk_signal_connect(GTK_OBJECT(options_button),
                    "clicked",
                    GTK_SIGNAL_FUNC (options_button_cb),
                    NULL);
gtk_signal_connect(GTK_OBJECT(fetch_button),
                    "clicked",
                    GTK_SIGNAL_FUNC (fetch_button_cb),
                    NULL);
gtk_signal_connect(GTK_OBJECT(compose_button),
                    "clicked",
                    GTK_SIGNAL_FUNC (compose_button_cb),
                    NULL);
gtk_signal_connect(GTK_OBJECT(delete_button),"
                    clicked",
                    GTK_SIGNAL_FUNC (delete_button_cb),
                    NULL);
```

（2）MailBox 管理

在 Email 应用中要用到 inbox，draftbox, sendbox 等等一些邮箱，其中 inbox 用于存储接收的邮件，draftbox 是草稿箱，sendbox 是发件箱。

由于各个邮箱的作用各不相同，而且在同一时间也只能查看一个邮箱。因此在打开一个邮箱的时候必须关闭其他的邮箱。其处理过程如图 5.5。

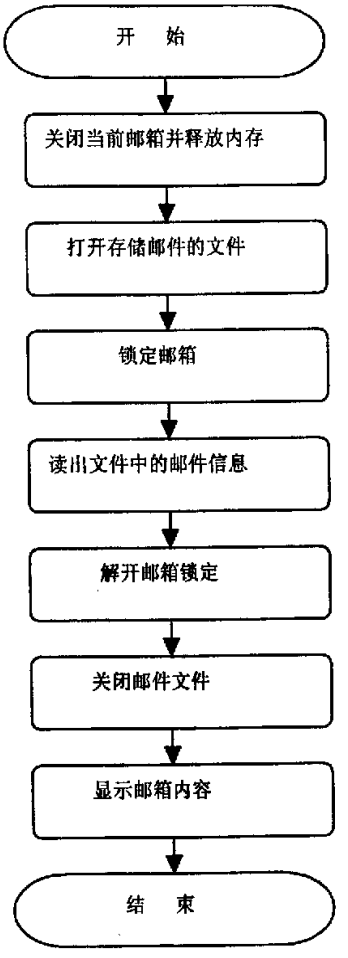

图 5.5 读取邮箱邮件的过程
Fig.5.5 Process of reading mail in mailbox

第六章 总结和展望

本章首先对本文工作进行总结，指出目前已完成的工作和有待完成的工作，最后对未来的工作和研究方向进行了展望。

6.1 本文工作总结

本文是在一个嵌入式系统的模拟器上实现 TCP/IP 应用层协议，详细的介绍了 Email 的发送和接收模块的设计和实现以及 Chat Mail 的发送和接收模块的设计和实现。并设计和实现了功能上基本满足系统设计目标的 Email 服务器和 Chat Mail 服务器。同时充分利用设计与实现分开的思想，把与执行环境有关的应用程序接口封装起来，使整个系统易于扩充和移植。最后并在嵌入式 Linux 系统下移植了一个邮件系统。

作为一个嵌入式系统模块的开发，必须保证本模块的可靠性，执行速度和可移植性。作者在开发此系统模块的过程中，切实的遵循软件工程的思想，并严格按照软件工程所要求的步骤，分阶段的完成此系统模块的设计和实现。作者在论文期间主要完成以下几方面的工作：

（1）源嵌入式系统的分析和总结。这是在此系统模块设计初期的主要工作，由于要实现的模块是在模拟器上应用，所以首先要分析模拟器的系统进行详细分析。同时由于要应用 Socket 通信编程，也就必须对原系统中中间层进行分析，已得到原系统对外通信的接口和通信的过程。独立编写了《模拟器分析报告》，并画出 Email 实现相关的类图，顺序图和状态图等

（2）资料的收集和分析。在对原系统有了一个初步的了解之后，对将要实现的模块有了一个初步的了解，接下来的工作就是准备相关资料。由于本模块需要使用大量的 RFC 标准，在实现之前必须对 RFC 标准有一个充分的理解，所以要从一些国际标准组织的网上下载一些 RFC 标准的源文件，并对其进行详细的分析和理解。

（3）Email 功能模块的设计。Email 功能是本论文的主体部分，它的合理设计始终贯穿在整个模块设计和实现的过程之中。在此期间，不但设计了本模块的类图，交互图。还对各个命令的活动情况进行了设计，得到各个命令的活动图。

（4）Email 功能模块的编码。在对 Email 功能模块的设计完成之后，就是编码实现设计中提出的功能。该模块的实现是本文的关键，对于编写的代码，本文作者经过反复的测试、修改、再测试，最终完成了此功能模块。

（5）Chat Mail 功能模块的设计和实现。在实现了 Email 功能模块之后，实现

Chat Mail 功能模块就比较容易了，因为他们的设计和实现的过程非常的相似。首先要设计出各个子功能模块的交互图。之后就是编码实现各个子功能模块。同样经过严格的测试之后，实现了此功能模块。

（6）嵌入式 Linux 下邮件功能的移植。

总之，本文反映了作者在嵌入式系统模拟器上实现 TCP/IP 协议上所做的工作。测试运行的结果初步证明上述功能模块的设计思路和可行的。但是，如果要真正投入到实际应用当中去，通常还应更多的考虑运行效率，运行的效率和可移植性。本文提到的各个模块都是在模拟器系统上进行开发的，由于时间和作者水平的限制，并未对上述问题做深入的考虑，希望读者批评指正。

6.2 对未来工作的展望

根据用户对嵌入式系统的 TCP/IP 协议簇应用的可扩展性、安全性及可管理性等要求，嵌入式系统 TCP/IP 应用层协议的实现形成了以下的发展趋势：

① 体系结构层次化，分布式处理，可移植性和可扩展性，便于功能添加。

② 支持多平台，使系统集成具有极大的灵活性。

③ 维护升级不必停止服务，保证服务的连续性和可用性。

④ 针对不同客户需求提供不同等级和类型的服务。

- 展望一 ：开发利用数字签名的安全通信系统

本文所论述的 Email 和 Chat Mail 功能模块对安全方面的考虑比较少，在当前应用中，各种各样的病毒以及黑客活动日益猖獗，在实际应用过程中，这些系统模块面临的安全问题就非常严峻。开发运用数字签名的安全通信系统，可以有效的防止病毒和黑客的侵扰，保证用户数据的安全。

- 展望二：上网浏览功能的实现

本文所论述的功能模块主要实现了电子邮件和聊天的功能。而在实际应用中，上网浏览也是一项非常常用的功能，实现这项功能也是非常有必要的。

参考文献

1. Dave Wood David Wood, Mark Stone. Programming Internet Email [M], Sevastopol: oreilly, 1999, 17-66
2. M. Tim Jones. TCP/IP Application Layer Protocols for Embedded Systems [M], Berkeley: SYBEX, 2002, 1-157
3. C. Kalt. Internet Relay Chat: Architecture RFC2810, 2000
4. Michael H. Jang, Michael Jang. Mastering Red Hat Linux 9 [M], Berkeley : SYBEX. 2002, 469-512
5. Bjarne Stroustrup. C++ Programming Language The Special Edition [M], Addison Wesley Professional, 2000, 67-89
6. J. Myers. SMTP Service Extension for Authentication, RFC2554, 1999
7. J. Klensin, R. Catoe P. Krumviede. IMAP/POP AUTHorize Extension for Simple Challenge/Response RFC 2195, 1997
8. David H. Crocker. Standard for the format of ARPA Internet text messages RFC 822, 1982
9. R. Rivest. The MD5 Message-Digest Algorithm RFC 1321, 1992
10. W. Simpson. The Point-to-Point Protocol (PPP), RFC1661, 1994
11. Jonathan B. Postel. Simple Mail Transfer Protocol RFC821, 1982
12. J. Klensin, M. Rose, N. Freed, E. Stefferud, D. Crocker. SMTP Service Extensions RFC1869, 1995
13. J. Myers, Carnegie Mellon, M. Rose. Post Office Protocol - Version 3, RFC1939, 1996
14. M. Crispin. Internet Message Access Protocol – Version 4rev1, RFC 2060, 1996
15. N. Freed, N. Borenstein. MIME: Mechanisms for Specifying and Describing the Format of Internet Message Bodies, RFC1341, 1992
16. N. Freed, N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, RFC2045, 1996
17. N. Freed, N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, RFC2046, 1996
18. J. Oikarinen, D. Reed. Internet Relay Chat Protocol, RFC1459, 1993
19. 康健. CDMA 移动通信系统中的数据互通 [J], 移动通信, 2000, (5): 42-44
20. Grady Booch, James Rumbaugh, Ivar Jacobson. UML 用户指南 [M], 北京: 机械工业出版社, 2001,139-174

21. Douglas E. Comer. 用 TCP/IP 进行网际互联 第一卷：原理、协议与结构 [M], 北京：电子工业出版社, 2003, 1-11,122-135, 287-307, 357-367
22. 王罡 林立志. 基于 Windows 的 TCP/IP 编程 北京:清华大学出版社 [M], 2002, 28-51, 348-362
23. Kurt Wall 著. 张辉 译. GNU/Linux 编程指南[M], 北京：清华大学出版社, 2002,495-519
24. 邹思轶. 嵌入式 Linux 设计与应用 [M], 北京：清华大学出版社,2002, 18-55
25. 张毅，赵国锋. 嵌入式 Internet 的几种接入方式比较 [J], 重庆邮电学院学报, 2002, 1(4) 12-13
26. Wayne Wolf 著. 孙玉芳 梁彬等译. 嵌入式计算系统设计原理[M], 北京：机械工业出版社,19-25,65-80
27. 师明珠. 嵌入式应用系统软件设计技术研究 [J], 计算机工程与应用, 2000, 38（7）：127-129
28. 刘晓. IMAP4 协议疑难解惑 [J]，中国电脑教育报，2000，5：35
29. 应用 GTK+编程，http://tech.ccidnet.com/pub/article/c303_a44259_p1.html
30. 嵌入式系统的特点，http://www0.ccidnet.com/tech/os/2001/08/23/58_3072.html

致　谢

本文是在导师张立东副教授的悉心指导和严格要求下完成的。我衷心感谢张老师两年半来在学业上的精心培养和在生活上无微不至的关怀与帮助，使我能顺利地完成硕士论文。张老师渊博的知识，认真求实和严谨治学的作风，忘我的工作态度和崇高的敬业精神，使我受益匪浅，终生难忘。

衷心的感谢东软集团中间件技术分公司嵌入式软件事业部的陈锡民部长和赵明老师。感谢你们在我做毕业论文的一年多的时间里给予我的无微不至的关怀和指导。还有公司的卢文龙老师，李昌忠老师，王晓峰老师，小白、候英启，张春海，我的搭档张敏丽小姐以及各位同事都给予了极大的帮助和支持。

另外，我要向同门好友洪波、王兴强、张赞伟、于鹏飞等表示由衷的谢意！感谢他们对我学习上的帮助和启发。也向一切给予我帮助，使我顺利完成硕士研究生学业的人表示真挚的谢意。

感谢一直支持和爱护我的父母和家人，无论在什么时候，他们总在我的背后默默的支持我，给我足够的勇气和信心让我去面对人生每一次挑战。