摘　要

随着计算机软、硬件技术的发展，特别是网络应用的不断普及，嵌入式应用在社会的各个领域越来越广泛和重要。作为嵌入式应用的基础，操作系统的优劣直接影响了系统的性能、开发和应用。因此，嵌入式操作系统是目前嵌入式系统研究的热点所在。由于 Linux 操作系统的开放源码特性，国内外越来越多的人员开始研究如何将 Linux 应用于嵌入式系统。但是由于 Linux 并不是针对嵌入式应用而设计的，所以在 Linux 应用于嵌入式系统的过程中，需要解决一些嵌入式系统所特有的问题。

由于应用领域的差异，嵌入式操作系统与一般操作系统不同，嵌入式应用通常对实时性能有较高的要求，同时由于嵌入式应用对体积有一定的限制，要求设备能耗低、体积小、重量轻。这些特性都对嵌入式操作系统提出了更高的要求。本文根据嵌入式应用的需求设计了一个基于 Linux 的嵌入式操作系统(Linux Based Embedded Operating System，简称 LEOS)，使之可以满足多数嵌入式应用在功能和效率上的需求，并在 Intel 公司的 StrongARM 嵌入式平台上实现了 LEOS。

在综合分析嵌入式系统特性和多种商用嵌入式操作系统的基础上，本文先从总体介绍了现有嵌入式操作系统的特性和原理，分析 Linux 作为嵌入式操作系统的优缺点，研究 Linux 应用于嵌入式领域的关键技术：内核实时化、文件系统和内核中文化。在此基础上，设计了实时化的双内核模式，提出混合式文件系统结构，并在分析 Linux 字符显示原理的基础上，给出内核中文化方案。最后阐述 LEOS 在 StrongARM 硬件平台上的具体设计实现和针对硬件平台的优化，并给出 LEOS 在 StrongARM 硬件平台上的性能测试和评估。

本文的研究成果可以应用于嵌入式开发中，为开发工作提供操作系统层次的支持。并且积累了嵌入式开发的经验，为后期嵌入式操作系统的改进和提高提供基础。

关键词： 嵌入式 操作系统 Linux StrongARM 实时 文件系统

Abstract

A Linux based Embedded Operating System for Intel® StrongARM™

Wei Wang (Computer Software and Theory)
Directed by Professor Qing Wang

In recent years, due to the phenomenal growth of networking and communication systems, embedded system becomes increasingly important on the Internet and other technical areas. As the base of embedded system, embedded operating system has a significant effect on the system performance. Consquently, embedded operating system has been one of the focuses of embedded system research. As an opensource operating system, Linux is more and more concerned for its well performance. However, as Linux is not designed for embedded system, many problemes should be resolved to port Linux for embedded system.

As the embedded applications become more and more complex, the operating system support and development environment become crucial. Furthermore, embedded systems generally need to run in a real-time environment and require the devices with small volume, light weight and low power consumption. All these characteristics make the design of embedded operating systems different and even more difficult compared with the general operating systems.In this thesis, after investigating other embedded operating systems, a Linux based embedded operating systems (LEOS) is proposed. And it is implemented on Intel® StrongARM™ embedded platform, which can support various embedded devices of the platform and satisfy the requirements of most embedded applications.

With analysis on the characteristics of embedded system and commercial embedded operating system, the properties and principles of embedded operating system are introduced first. The advantages and disadvantages for Linux to be embedded operating system are also analyzed. Secondly, the core techniques in embedded Linux are studied e.g. the real-time characteristic of the kernel, the blended file system and the Chinese support. Based on this, the design of double kernel model for real-time, blended file system and Chinese support in kernel is proposed. The development and optimization of an embedded Linux implemented on Intel® StrongARM™ platform is studied in detail. Finally, the system test and evaluation of LEOS on StrongARM™ is presented.

The research of this thesis can be used in the development of embedded applications. The resolution and optimization on Intel® StrongARM™ platform is an exploration of using Linux for embedded system.

Key Words: Embedded, Operating System, Linux, StrongARM, Real-time, File System

独创性声明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得中科院软件研究所或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所作的任何贡献均已在论文中作了明确的说明并表示了谢意。

本人签名：_____ 日期 _____

关于论文知识产权与使用授权的说明

本人完全了解中科院软件研究所有关保留和使用学位论文的规定，即：论文知识产权属软件所所有；软件所有权保留送交论文的复印件，允许查阅和借阅论文；软件所可以公布论文的全部或部分内容，可以允许采用影印、缩印或其他复制手段保存论文。（保密的论文在解密后遵守此规定）

本人签名：_____ 日期_____

导师签名：_____ 日期_____

第一章 绪论

随着计算机应用的普及、互联网技术的发展以及纳米微电子技术的突破，计算机正有力推动着21世纪工业生产、商业活动、科学实验和家庭生活等领域自动化和信息化进程。计算机技术在各个领域的应用为嵌入式系统的发展提供了崭新而巨大的空间，各种网络设备、机器人、家用电子系统以及信息家电等器材设备中，嵌入式系统的应用越来越广泛。

根据英国电机工程师协会的定义：嵌入式系统为控制、监视或辅助设备、机器或甚至工厂运作的装置[1]。嵌入式系统是一种电脑软件与硬件的综合体，强调“量身定做”的原则，也就是基于某一种特殊用途的嵌入式系统，必须针对这项用途开发出截然不同的一项系统。

1.1 嵌入式系统研究现状及发展趋势

整个嵌入式系统的发展历史相当悠久，可以追溯至1971年由Intel公司推出有史以来第一颗微处理器4004开始，而微处理器的成功也在接下来的二十年里改变了人类的生活，某些典型的嵌入式系统几乎让人感觉不到它的存在，包括生活中常见的微波炉、冷气机、电冰箱等等，不过近年来新兴的嵌入式系统领域发展相当快速，目前嵌入式系统产业中主要发展的技术热点包括嵌入式操作系统、系统芯片设计、应用软件发展以及内容服务这些方面[2][3][4]：

1、嵌入式操作系统

与PC操作系统(WIN2000/NT)比较，嵌入式操作系统不要求全能，但必须能够依据系统设计规格，高效的发挥出硬件的运算能力，使得产品达到效率价格比的最佳化，大多数的嵌入式系统要求全自动完成所设定的工作，例如工厂或是银行的系统。除了原本在嵌入式领域占有重要地位的VxWORKS、QNX等操作系统之外，新兴的产品包括Palm OS、Windows CE、Linux等，其中基于Linux的嵌入式操作系统免费授权的特性，已为数家国际知名厂商所用。

2、系统芯片(System on Chip,以下简称SoC)

嵌入式产品所需的处理器及芯片组相对PC要求体积小、散热佳、省电，因此多用高整合度的SoC为其处理器核心，为了加速缩小科技进步与设计生产能力间的差距，加速SOC的实现，SIP(Silicon Intellectual Property)即所谓硅智慧财产权的重使用，成为各方瞩目的焦点。

这种类型的产品众多，例如国家半导体(NS)的Geode SC1400整合CPU、绘图芯片、MPEG-2、I/O及TV out等功能，就适合于家庭数码影音设备的产品运用。其它则还有Intel、Motorola、Transmeta等厂商投入单芯片的设计。

3、应用软件

嵌入式软件可区分为用户端的应用软件及服务器端的整合软件，服务器端的软件可能以Linux或是Windows为核心，搭配各种数据库系统，用户端由于各种产品种类繁多，可开发出的软件也相对增加，例如Palm据称有上万种应用软件可以使用。除了原本各种平台专属的应用软件之外，现在新型的嵌入式系统通常会支持Java，利用其跨平台的特性，使得可用软件的种类变得更多。

4、服务

由于嵌入式产品能随身携带或走入居家生活，故其一方面体积上要求轻薄短小、造型及颜色个人化、输入自然化、输出多媒体化，才能吸引消费者。另一方面由于嵌入式产品与网络关系紧密，所以与网络服务提供者或电子商务业者极易结合，也就是嵌入式产品连上网络的入口网站及其内容(HTML/XML)可能由厂商负责提供，比如日本 NTT DoCoMo 所发展的 iMode 服务，该项服务在日本已经获得巨大成功。

操作系统作为计算机产品的基础，也是嵌入式系统的基础，嵌入式操作系统决定了其上的嵌入式应用的功能和效率，也正在成为目前研究的热点，相关的成果层出不穷。

从八十年代起，国际上就开始进行商用嵌入式操作系统和专有操作系统的开发[5][6]。一些嵌入式操作系统应用范围很广泛，其中著名的嵌入式操作系统有：

◆ Windows CE：

Microsoft Windows CE 是一个简洁的，高效率的多平台操作系统。它是为有限资源的平台设计的多线程，完整优先权，多任务的操作系统。Windows CE 的模块化设计允许它对于从掌上电脑到专用的工业控制器的用户电子设备进行定制。Windows CE 的基本内核需要至少 200K 的 ROM。从 SEGA 的 DreamCast 游戏机到现在大部分的高价掌上电脑，都采用了 Windows CE，它的缺点是价格太高，容易使得整个产品的成本急剧上升，影响了它在国内的发展。

◆ VxWorks：

VxWorks 是目前嵌入式系统领域中使用最广泛，市场占有率最高的系统。它支持多种处理器，如 x86，i960，Sun Sparc，Motorola MC68xxx，MIPS RX000，POWER PC 等等。使用的是和 UNIX 不兼容的环境，大多数的 VxWorks API 是专有的。采用 GNU 的编译和调试器。VxWorks 由于拥有很高的实时性能，应用领域非常广泛，但缺点也是价格太高。

◆ pSOS：

pSOS 是一个模块化，高性能的实时操作系统，专为嵌入式微处理器设计，提供一个完全多任务环境，在定制的或是商业化的硬件上提供高性能和高可靠性。pSOS 允许开发者将操作系统的功能和内存需求定制成每一个应用所需的系统。开发者可以利用它来实现从简单的单个独立设备到复杂的、网络化的多处理器系统。

◆ QNX：

QNX 是一个实时的，可扩充的操作系统，它遵循 POSIX.1 (标准程序接口)和 POSIX.2 (Shell 和工具)、部分遵循 POSIX.1b(实时扩展)。它提供了一个很小的微内核以及一些可选的配合进程。其内核仅提供 4 种服务：进程调度、进程间通信、底层网络通信和中断处理，其进程在独立的地址空间运行。所有其它 OS 服务都实现为协作的用户进程，因此 QNX 内核非常小巧(QNX4.x 大约为 12Kb)而且运行速度极快。这个灵活的结构可以使用户根据实际的需求将系统配置成微小的嵌入式操作系统或是包括几百个处理器的超级虚拟机操作系统。

◆ Palm OS：

Palm 公司的 Palm OS 在 PDA 市场上占有很大的市场份额，它有开放的操作系统应用程序接口（API），开发商可以根据需要自行开发所需要的应用程序。目前已经有总共 3500 多个应用程序可以运行在 Palm 平台上，其中大部分应用程序均为其他厂商和个人所开发，使得 Palm Pilot 的功能得以不断增多。这些软件包括计算器、各种游戏、电子宠物、地理信息等等。在开发环境方面，可以在 Windows

2000，Windows NT 以及 Macintosh 下安装 Palm Pilot Desktop，Plam 可以与流行的 PC 平台上的应用程序如 Word，Excel 等进行数据交换。

◆ OS-9：

Microwave 的 OS-9 是为微处理器关键实时任务而设计的操作系统，广泛应用于高科技产品中，包括消费电子产品，工业自动化，无线通讯产品，医疗仪器，数字电视/多媒体设备中。它提供了很好的安全性和容错性。与其他的嵌入式系统相比，它的灵活性和可升级性非常突出。

◆ LynxOS：

Lynx Real-time Systems 的 LynxOS 是一个分布式、嵌入式、可规模扩展的实时操作系统，它遵循 POSIX.1a、POSIX.1b 和 POSIX.1c 标准。LynxOS 支持线程概念，提供 256 个全局用户线程优先级；提供一些传统的，非实时系统的服务特征；包括基于调用需求的虚拟内存，一个基于 Motif 的用户图形界面，与工业标准兼容的网络系统以及应用开发工具。

这些系统均为商业嵌入式操作系统，虽然它们能够为嵌入式应用提供较为全面的支持，但由于价格高昂，源码封闭等原因，使得其应用受到限制，而且对于某些特殊应用，如国防应用、科研开发等，这些商业嵌入式操作系统并不适合。而 Linux 以其开放源码等特性，使得它能够在嵌入式操作系统占有一席之地，并且正在为越来越多的人所关注，通过 Internet，大量的研究人员正在改进 Linux，新技术很快会被应用到实践中，这些都使得基于 Linux 的嵌入式操作系统成为嵌入式领域的一个重要研究内容。

然而，由于 Linux 设计初衷并不是应用于嵌入式系统，将 Linux 作为做为嵌入式操作系统还有许多问题尚待解决：

- Linux 没有提供实时机制，对于实时任务 Linux 需要添加实时模块。而这些模块运行的内核空间正是操作系统实现调度策略、硬件中断和执行程序的部分。这些实时软件模块是在内核空间运行的，因此代码错误可能会破坏操作系统从而影响整个系统的可靠性，这对于实时应用是一个非常严重的弱点。
- 嵌入式系统对体积和能耗有较高要求，通常使用 Flash 做为永久存储器，由于嵌入式电源的不稳定性，文件系统必须针对 Flash 进行相应的优化和改进，并针对嵌入式系统的特点，提高文件系统的稳定性和执行效率，充分发挥硬件性能。
- Linux 不支持双字节显示，如果使用通常的外挂式中文显示，又会占用过多的系统资源。对于嵌入式应用，需要在内核层支持中文显示。
- 嵌入式应用会使用到很多不同的设备和接口，Linux 在这方面也需要针对不同的平台开发相应的驱动。
- 嵌入式设备处理能力不高，Linux 的 X-Windows 图形界面如果直接使用在嵌入式系统中，会加大系统的处理负担，甚至可能完全不能使用。需要为嵌入式系统提供体积小、效率高、适合小型应用的图形界面系统。

1.2 论文的工作

本文针对基于 Linux 的嵌入式操作系统的现状，研究如何为嵌入式应用提供完整的操作系统解决方案，解决将 Linux 应用于嵌入式所面临的问题和技术难

点，设计一个基于 Linux 的嵌入式操作系统，并在 StrongARM 平台上做具体设计、实现和性能评估，解决了实现中碰到的平台相关问题，其中主要工作包括：

- 在分析嵌入式应用特性的基础上，总结出将 Linux 应用于嵌入式操作系统所面临的问题、必须的改进及其对应的关键技术：内核实时化、文件系统、内核中文化。
- 针对基于 Linux 的嵌入式操作系统的关键技术提出相应的设计和优化方案，包括双内核的实时化设计、混合式文件系统设计和内核的中文化设计。
- 在 StrongARM 平台上设计并实现了基于 Linux 的嵌入式操作系统，解决了具体实现过程中的问题。使得嵌入式操作系统能够为嵌入式应用提供较为完整的底层支持，成为嵌入式开发的基础。

本文的研究工作是为嵌入式开发的基础性工作，解决了嵌入式系统中操作系统的问题，为嵌入式应用提供了操作系统层次的支持和保证。

1.3 论文的组织

本文第一章“绪论”分析了嵌入式系统的研究现状及发展趋势，并介绍了论文的工作和组织结构。

第二章“基于 Linux 的嵌入式操作系统综述及工作背景”介绍 Linux 在嵌入式系统方面的应用现状及其优缺点，主要包括基于 Linux 的嵌入式操作系统的主流产品，Linux 对嵌入式芯片的支持，分析 Linux 在嵌入式应用方面的优势与不足，并概述了本文的实现环境：StrongARM 硬件平台。

第三章“基于 Linux 的嵌入式操作系统关键技术研究”分析构建基于 Linux 的嵌入式操作系统的关键技术，针对这些问题，提出相应的设计方案，主要包括双内核的实时化设计、混合式文件系统设计和内核中文化设计。

第四章“StongArm 平台上 LEOS 的设计与实现”描述基于 Linux 的嵌入式实时操作系统在 StrongARM 平台上的设计与实现，以及相关的优化和改进，并对系统性能和功能进行了测试和分析比较，对系统进行了评估。

最后对论文的工作进行了总结，介绍系统可能的应用领域和进一步的工作。

第二章 基于 Linux 的嵌入式操作系统综述及工作背景

随着 Linux 和嵌入式应用的不断发展，越来越多的人关注于如何将二者进行结合，产生了基于 Linux 的嵌入式操作系统。本章概述了 Linux 及基于 Linux 的嵌入式操作系统发展现状，分析了 Linux 应用于嵌入式的优点介绍了 Linux 所支持的硬件平台和本文研究工作的目标环境：StrongARM 硬件平台。

2.1 Linux 的发展现状

Linux[7]是一个遵循 POSIX 标准的免费操作系统，具有 BSD 和 SYS V 的扩展特性，其在外表和性能上同常见的 UNIX 非常相象，但是所有系统内核代码已经全部被重新编写了。它的版权所有者是芬兰籍的 Linus B. Torvalds 先生和其他开发人员，并且遵循通用公共许可证（GNU General Public License，以下简称 GPL）声明。

Linus 于 1991 年 10 月 5 日公开发表 Linux 的第一个版本：0.02 版，随后 Linux 以两个星期出一次新版本的速度迅速成长，并于 1994 年 3 月 14 日发表了它的第一个正式版本：1.0，由于世界各地的研究人员不断的改进，使得 Linux 内核发展十分迅速，目前最新的内核稳定版本是 2.4.20，开发版本为 2.5.67。 伴随着 Linux 内核版本的不断更新，Linux 的发行版也不断完善。所谓发行版就是将 Linux 系统的内核与外围实用程序软件和文档包装起来，并提供一些系统安装界面和系统配置、设定与管理工具，组成一种发行版本。Linux 的各个发行版本，都是使用 Linus 主导开发并发布的同一个 Linux 内核，因此在内核层不存在兼容性问题。每个版本只是在发行版本的最外层才有所体现，而绝不是 Linux 本身特别是内核不统一或是不兼容。目前最流行的几个正式版本有：

- Slackware，是最早的 Linux 正式版本之一，它遵循 BSD 的风格，尤其是在系统启动脚本方面。现有的版本是 Slackware 9.0，基于 Linux 2.4.20 内核。
- Debian，是开放源代码的操作系统，目前基于 Linux 2.4 内核。它由许多志愿者维护，是真正的非商业化 Linux，现有最新的版本是 3.0r1。
- RedHat Linux，是 Linux 最早的商业版本之一。它在美国和其他英语国家市场上获得了较大的成功。现有的最新版本是 RedHat Linux 9，基于 Linux 2.4.20 内核。
- SuSE，由德国人开发，是在欧洲大陆最流行的版本之一。现有最新版本是 SuSE 8.2，基于 Linux 2.4.20 内核。
- TurboLinux 公司是以推出高性能服务器而著称的 Linux 厂商。它是亚洲占市场最大的商业版本，在中国、日本和韩国都取得了巨大的成功。现在发行的版本是 TurboLinux 8，基于 Linux 2.4.18 内核。

目前 Linux 除了针对 PC 和服务器的版本外，随着嵌入式应用的发展，也出现了大量的基于 Linux 的嵌入式操作系统，通过修改 Linux 内核，再配套相应硬件平台的编译器和一些常用工具集，其高层处理相对于普通 Linux 基本没有改变，大量 Linux 下的应用程序都可以运行于嵌入式系统之上。由于 Linux 的免费、兼容等特性，基于 Linux 的嵌入式操作系统的发展非常迅速，逐渐能够在嵌入式

领域获得一席之地，并正在成为新的嵌入式研究热点，将 Linux 发展为一个嵌入式操作系统具有很多优点[8]：

◆ 系统稳定、功能强大、支持多种硬件平台、应用软件多

Linux 在许多方面与 UNIX 类似，但它是一个完全独立的操作系统，它可以非常稳定地运行在多种体系结构的处理器上。最新的 Linux 内核支持 Intel x86、Motorola/IBM PowerPC、Compaq(DEC)Alpha、IA 64、S/390、SuperH 等微处理器体系结构。

Linux 操作系统本身的内核体系结构相当简单。网络和文件系统以模块形式置于内核的上层。驱动程序和其它部件可在运行时作为可加载模块编译到或者是添加到内核中。这为构造可定制的嵌入式系统提供了高度模块化的构造方法。在许多应用情况下系统需要定制的驱动程序和应用程序以提供附加功能。

Linux 的系统界面和编程接口和传统的 UNIX 类似，UNIX 下的程序员可以很方便的从 UNIX 环境转移到 Linux 环境下来。而不像从 UNIX 环境转移到 Windows 开发环境那样复杂。

在 Linux 平台上的应用软件也不断得到扩充。许多著名的商业软件都有了 Linux 下的版本：Applix 公司和 Star 公司提供了多种字处理、电子表格、图形处理的应用软件；Corel WordPerfect、Oracle 数据库、Netscape Navigator 网络浏览器、Apache 网络服务器、Adobe Acrobat Reader 等等 Linux 下的应用程序都已经推出。

在网络服务器市场上，近几年商用 UNIX 系统在往大而复杂的方向发展，使得 UNIX 的复杂性不断增加，管理整个 UNIX 系统也就变得越来越复杂。Linux 简单易用，系统管理也比较容易上手，从而成为在服务器高端的一个重要选择，并且有不断上升的趋势，大有取代昂贵、复杂的商用 UNIX 的趋势。

◆ 使用成本低

所有的商业用操作系统如 Microsoft 公司的 Windows CE 系列，都需要为每一个拷贝支付相当数量的费用。在其下的应用软件每一个都需要大量的支出来获得。商用操作系统下建立一个开发工具链，除了要为操作系统本身付费之外，还要为组成工具链的应用软件工具包支付大量的使用费用。但是 Linux 是免费软件，只要遵守 GPL 的规定，就可以免费获得拷贝。Linux 下有同样遵循 GPL 规定的 C、C++、Java 等等一系列的软件工具开发包，从功能角度上看并不亚于商用开发包，同时可以极大的降低开发成本。这点优势是其他商用操作系统无法比拟的。

◆ 文档完善

Linux 有非常多的文档支持，从为初学者准备的各种教程到非常详细的联机帮助文档。Linux 是互联网充分发展的产物，许多关于 Linux 的文档都可以在 Internet 上找到和下载。Linux 文档项目(Linux Document Project)是为 Linux 提供系统化的文档支持的项目，在世界上许多程序员和用户的帮助下，该项目已经收集了非常详细的系统文档和使用文档。

◆ 强大的网络功能

Linux 操作系统最突出的是网络部分，基本上所有的网络协议和网络接口都在 Linux 中实现，Linux 内核能够比标准的 UNIX 更加高效地处理网络协议，系统的网络吞吐性能非常好，这也是为什么 Linux 在网络服务器市场上占据越来越大市场份额的一个原因。

当然，最重要的是 Linux 不是某个公司的私有财产，它是一个开放源码软件，

是免费和源代码公开的。Linux 在这几年不断成熟，越来越多的人开始使用 Linux，以前 Linux 只是黑客和专家所使用的操作系统，而现在即使是电脑的普通用户也在使用 Linux。为 Linux 提供服务的公司也越来越多，为客户提供专业化的技术支持。Linux 有一个庞大的支持开发者群体，他们编写驱动程序和其它的更新程序并且免费通过 Internet 网络进行分发。对于新硬件，Linux 驱动程序有时甚至比用于 UNIX 系统如 Solaris 的驱动程序还来得及时。Linux 的庞大的志愿者网络在生产补丁程序方面反应也很快。如，当 Pentium II 的 bug（97 年 Pentium II 处理器的微指令发现设计问题）被发现以后，Linux 就是最早提供解决这个问题的方案的操作系统。如果一个 Linux 应用程序流行起来，用户一般都可以通过 Linux 新闻组得到很好的支持。有很多 Usenet 新闻组可供 Linux 用户寻求帮助。对一般 Linux 问题的回答时间可同一些厂商的 E-mail 支持相比。对 Linux 的支持绝大部分是通过用户团体在 Usenet 新闻组上提供的。上面广泛收集有大量的 FAQ，其内容包括 Linux 安装、配置和故障定位等各个方面。在 Usenet 上提供的许多材料已经被一些出版商如 Walnus Creek 等公开出版了。所有的这些，都是现有的其他嵌入式操作系统所无法比拟的。

2.2 基于 Linux 的嵌入式操作系统发展现状

基于 Linux 的嵌入式操作系统的各种优点和特性，使得基于 Linux 的嵌入式操作系统正在成为研究热点，目前已经有大量的成果和产品，其中比较突出的有：

◆ MontaVista Linux 2.1[9]

MontaVista Linux 2.1 是业界领先的基于 Linux 的嵌入式操作系统解决方案供应商 MontaVista 软件公司最新的下一代基于 Linux 的嵌入式操作系统操作平台。MontaVista Linux 前身就是应用很广的 Hard Hat Linux，最近被 MontaVista 公司改名为 MontaVista Linux。该最新版本广泛地支持各类嵌入式应用，为通信基础设施、网络、消费电子、仪表以及工控设备提供标准的基于 Linux 的嵌入式操作系统平台。

MontaVista Linux 2.1 专业版广泛地支持各类嵌入式处理器体系结构、CPU 板卡以及软件组件，包括 6 种体系结构的 20 款处理器，x86/IA-32、PowerPC、XScale、ARM、MIPS 以及 SH。MontaVista Linux 2.1 包括 KDevelop IDE、目标配置工具、库优化工具。另外，它还提供超过 215 个应用软件包。MontaVista Linux 2.1 基于最新的 Linux 2.4.17 稳定内核，提供支持 x86、MIPS、SH 以及 PowerPC 体系结构的实时抢占式内核。

◆ uClinux[10]

uClinux 是专为无存储器管理单元（MMU）的微控制器设计的基于 Linux 的嵌入式操作系统。uClinux 首先被移植到摩托罗拉的 MC68328 DragonBall 集成微处理器上，随后 uClinux 越来越受到业界的关注，被移植到更多的无 MMU 芯片上。第一个成功应用的目标系统是采用 TRG SuperPilot 板和专门为 Linux/PalmPilot 定制的引导程序的 3Com PalmPilot。

uClinux 已移植支持的微控制器和微处理器包括摩托罗拉 DragonBall、ARM7TDMI、MC68EN302、Axis ETRAX 、Intel i960、PRISMA、Atari 68k、ETRAX 等。

由于 uClinux 主要是针对无 MMU 微处理器开发的，因此，在 uClinux 上实现多任务功能则是一个非常棘手的问题。不过，大多数内核的二进制代码和源代

码都被重写，这进一步缩减了 uClinux 内核的代码。uClinux 的内核要比原 Linux 2.0 内核小的多，但保留了 Linux 操作系统的主要优点：稳定性，优异的网络能力以及优秀的文件系统支持。

◆ LynuxWorks BlueCat[11]

BlueCat Linux 是 LynuxWorks 公司的旗舰产品。该产品使用 2.4 内核，支持多种处理器，包括 Intel XScale、Intel IXP1200 等。 LynuxWorks 已经成为面向 Intel Exchange Architecture 的主要 Linux 软件供应商。LynuxWorks 公司推出的面向 XScale、IXP1200 以及嵌入式 Intel Architecture 的第二代 BlueCat 产品将缩短产品从研制到投放市场的时间。

由于 Linux 是针对桌面和服务器应用而设计的，所以当 Linux 应用于嵌入式系统时需要针对嵌入式应用的特点进行相应的改进，Linux 的这些嵌入式产品虽然解决了嵌入式应用中的很多问题，但很多公司已经开始收费。目前国内、国际上针对嵌入式的完整 Linux 解决方案很少免费公开，很多研究都是针对某一点进行的，整体的解决方案往往不够完整，而且在内核级别一般不支持中国用户所关心的中文显示问题。

2.3 Linux 支持的嵌入式硬件平台

与 PC 不同，嵌入式领域中存在上百种 CPU ，针对常用的嵌入式 CPU，如 X86、StrongARM、ARM、MIPS、POWER PC、SA1110、NEC VR4181、VR4121、MediaGX、ARM7 等，Linux 开放源码团队和相应的开发厂商为其中的一些 CPU 提供了开发方案，包括相应架构平台下的编译器、应用工具和各种启动工具等，可以说，就常用的嵌入式平台而言，使用 Linux 是最经济，也是最快速的办法之一。然而，由于 Linux 基于开放源码的自由团队开发的产品，没有资金支持，缺乏完整的质量保证体系，虽然通过互联网很多人在为 Linux 做改进，但仍然存在一些问题，在系统的兼容性、稳定性和易用性上 Linux 与商业嵌入式操作系统还有相当的差距。

Intel 公司的 StrongARM 平台[12]是专为嵌入式应用而设计，具有功能强大、能耗小和成本低的特点，能够适用于多种场合的嵌入式应用。针对 StrongARM 平台的交叉编译器也比较成熟[13]，目前已经有多家公司开发出的产品使用该处理器，如惠普的 Jornada 系列 PDA、iPAQ 系列 PDA 等，但这些产品使用的均是 Windows CE 或 PocketPC 操作系统，支持该硬件平台的 Linux 操作系统还不够完善，所以本文研究工作的实现采用了 StrongARM 硬件平台做为开发环境。

2.4 StrongARM 硬件平台介绍

StrongARM 是由 Intel 公司开发的一款面向嵌入式领域的处理器系列[14][15][16]，在综合考虑多种通用嵌入式硬件平台的基础上，由于 StrongARM 硬件平台对于本文的通用嵌入式操作系统能够提供很好的支持，并能够满足大多数应用，本文选取 StrongARM 硬件平台做为嵌入式操作系统开发的基础，它具有如下特性：

- ◆ StrongARM 是采用 RISC 技术的高性能、低功耗的处理芯片，它具有性能高、成本低和能耗省的特点，
- ◆ 适用于多种领域，如嵌入控制、消费/教育类多媒体、DSP 和移动式应用

等。

随着它的推出 Intel 公司也开发了相应的开发板，本论文的实现工作工作主要是基于 SA1110 开发板[17]（也称 Assabet）完成的，其逻辑图如 2-1 所示：

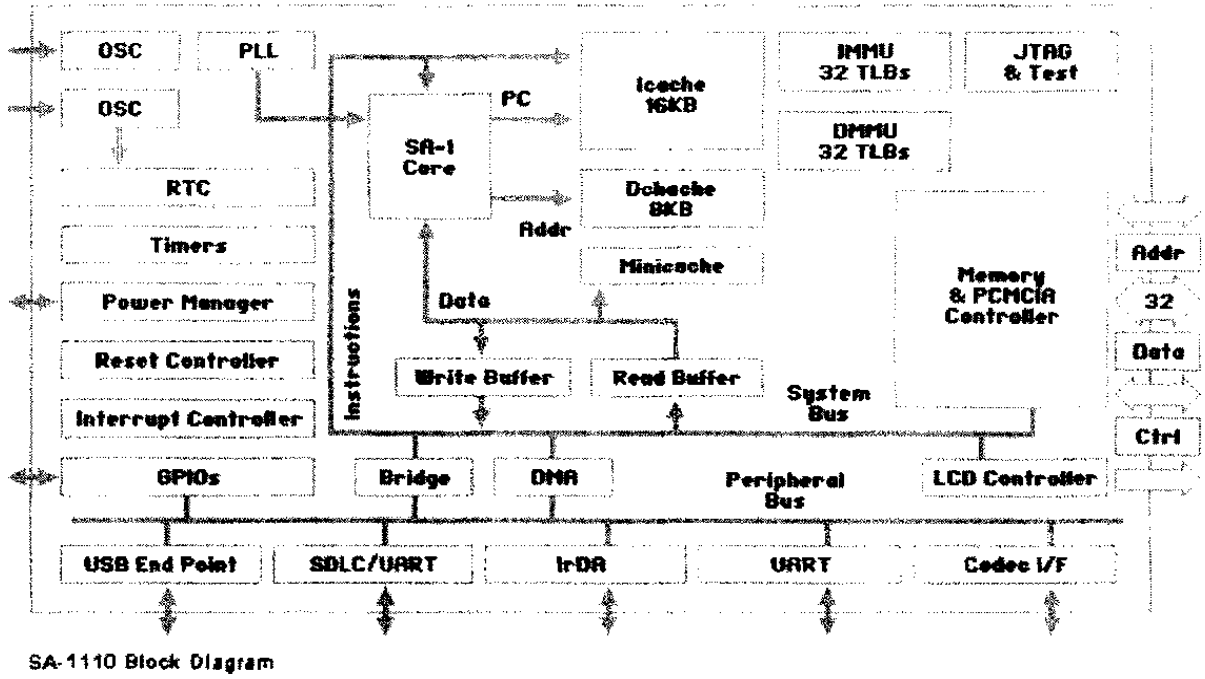

图 2-1 StrongARM 开发板逻辑图

该开发板不仅包含了 SA1110 处理器，还集成有 Flash 芯片、显示屏、触摸屏、串口、USB 接口、音频接口等，其硬件特性[18]如下：

- SA-1110 微处理器芯片，206MHz，内含 32bit 的 RISC 处理器；
- Flash Memory 容量 128M bit（最大为 256Mbit）；
- SDRAM 容量 128M bit100MHz（最大为 512Mbit）；
- 允许最大可能的 SDRAM 存储器带宽的无缓冲的主存储界面；
- 3.9″反射彩色 TFT LCD 触摸屏，背景光；
- UDA1341 立体译码器、支持麦克风、扬声器、POTS 线的软 modem DAA 连接、触摸屏和支持高质量的 16 位立体声音频录音和重放；
- 电池供电：采用一个锂电池通过高性能 DC-DC 转换器提供系统电源；
- CDMA，GSM，Bluetooth 模块的 RF 模块接口；
- 可供 JTAG 编程，RS232，电源输入和电话的基站连接器；
- 可通过 JTAG 或以太网口加载操作系统。

StrongARM 开发板的结构图如图 2-2 所示：

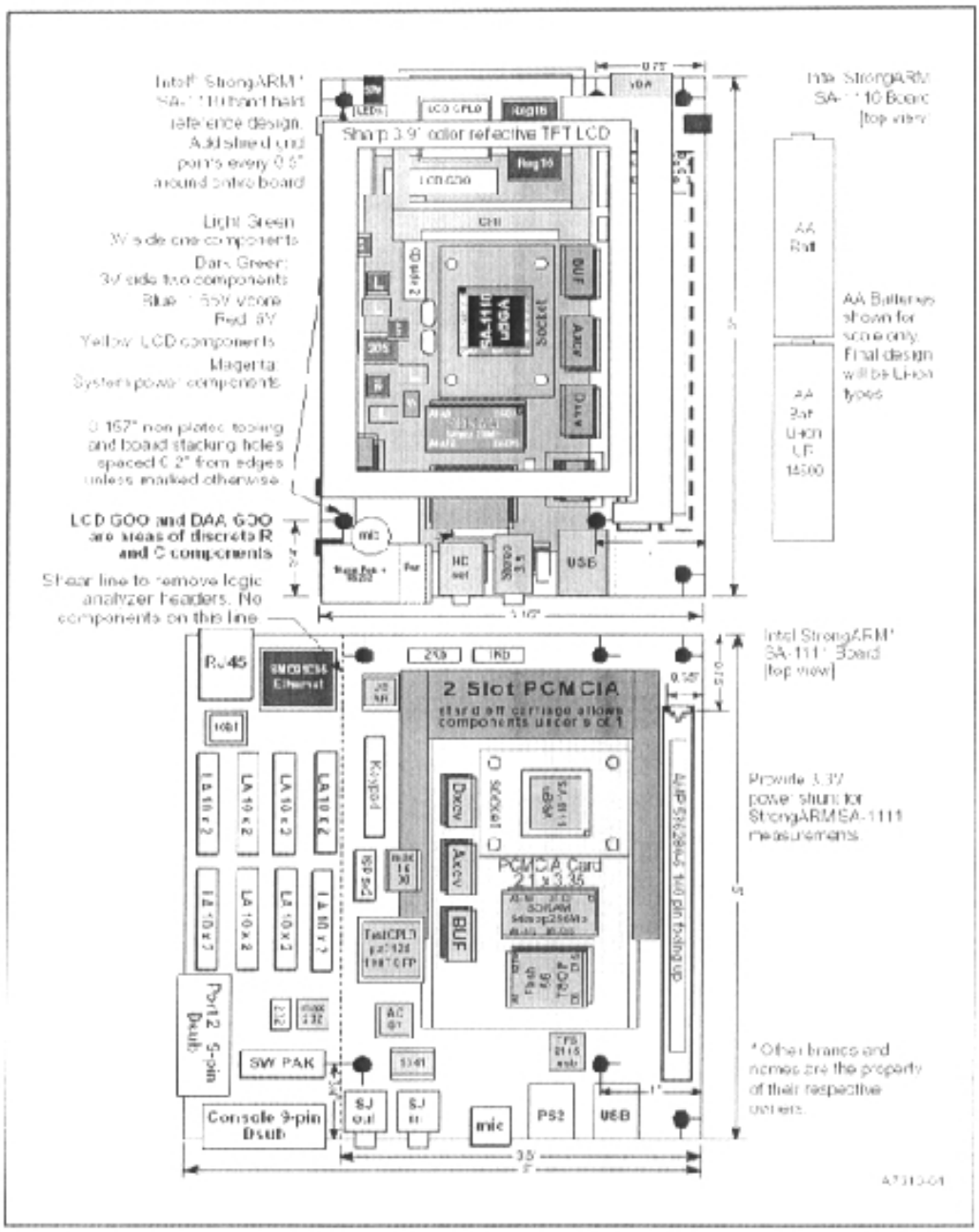

图 2-2 StrongARM 开发板结构图

开发板实物图如图 2-3 所示：

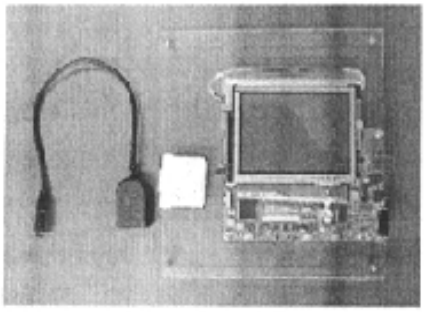

图 2-3 StrongARM 开发板实物图

2.5 小结

本章概述了 Linux 和基于 Linux 的嵌入式操作系统的发展现状，分析了 Linux 应用于嵌入式系统的优缺点，并介绍了本文实现部分的目标环境：StrongARM 硬件平台，本文的后续章节将详细讨论基于 Linux 的嵌入式操作系统的关键技术及其在 StrongARM 平台上的具体设计与实现。

第三章 基于 Linux 的嵌入式操作系统关键技术研究

本文的研究目标是为嵌入式系统应用设计开发相应的基于 Linux 的嵌入式操作系统，该操作系统需要为嵌入式应用和开发提供较为全面的操作系统底层支持。

将 Linux 应用于嵌入式系统主要面临的问题是如何使得 Linux 满足嵌入式应用的特点，如何在标准 Linux 操作系统的基础上，配合启动程序和文件系统，构建嵌入式操作系统。主要问题包括：

1、实时问题：很多领域如工业控制、航空设备、网络设备等嵌入式应用对于系统的响应时间和任务处理时间有严格的要求，这就要求操作系统在中断处理、任务调度等功能上提供相应的保证。Linux 内核的任务调度策略采用的是非抢占式的分时机制，难以满足实时要求。同时，Linux 对于中断的处理也不能够完全满足嵌入式系统的实时需求，在实时方面基于 Linux 的嵌入式操作系统需要进行改进，提供实时中断处理能力和实时调度策略。

2、文件系统问题：嵌入式系统的一个特点就是对系统体积的有很高的要求，使得 IDE 硬盘等体积较大的存储设备难以使用，目前嵌入式设备广泛使用闪存(Flash)设备为系统提供永久存储机制。某些手持嵌入式设备使用电池供电，电源的可靠性不强，这些都要求嵌入式操作系统所使用的文件系统能够在突然断电的情况下具有很好的稳定性和容错性，同时也需要针对 Flash 的存储特性，改进文件系统的操作。

3、中文问题：针对嵌入式应用的情况，中文用户希望操作系统支持中文的显示和处理，目前虽然有很多 Linux 中文化方案，一些 Linux 发行版也支持中文显示，但绝大多数都是在应用层进行中文化。而对于嵌入式设备，由于其处理能力和存储能力有限，更好的方式是在内核层进行汉化，提供内核级别的中文支持。

通过以上分析，基于 Linux 的嵌入式操作系统(Linux based Embedded Operating System,以下简称 LEOS)的关键技术的包括：为嵌入式实时应用设计系统级别的实时机制、实时终端、实时任务通信支持；针对嵌入式的特点设计嵌入式文件系统，提高系统性能；为嵌入式应用提供中文化的人机界面，支持中文显示。

本章主要研究 LEOS 的关键技术，提出了双内核实时化方案、混合式文件系统方案和内核中文化方案。与硬件相关的研究内容将在下一章进行讨论。

3.1 LEOS 内核实时化技术研究

在嵌入式系统应用中，很多情况下要求系统能够在一定时间内响应某个或多个请求，像嵌入式的工业控制系统，对任务的执行时间有很严格的限制，每个任务都必须在特定时间之前完成，否则就可能造成整个系统的混乱，这就要求操作系统能够提供实时支持，即操作系统应具有实时操作系统（Real Time Operating System）的特性。然而，标准 Linux 并不支持实时处理，它采用的是分时调度策略，中断处理也不够快速，在实时方面，Linux 需要进行改进。

3.1.1 实时操作系统特性

实时操作系统是指具有实时性，能支持实时控制系统工作的操作系统。首要任务是调度一切可利用的资源完成实时控制任务，其次才着眼于提高计算机系统的使用效率，主要特点是要满足对时间的限制和要求[19][20][21][22][23]。

实时多任务操作系统与通常的分时多任务操作系统(如 Linux、Windows)有明显的区别。具体的说，对于分时操作系统，软件的执行在时间上的要求，并不严格，时间上的错误，一般不会造成灾难性的后果。而对于实时操作系统，主要任务是对事件进行实时的处理，虽然事件可能在无法预知的时刻到达，但是软件上必须在事件发生时能够在严格的时限内做出响应（系统响应时间），即使是在尖峰负荷下，也应如此，系统时间响应的超时就可能导致预期工作的失败。另外，实时操作系统的主要特点是具有系统的可确定性，即系统能对运行情况的最好和最坏等的情况能做出精确的估计。

实时操作系统中的主要概念：

系统响应时间(System response time)是系统发出处理要求到系统给出应答信号的时间。

任务换道时间(Context-switching time)是任务之间切换而使用的时间。

中断延迟(Interrupt latency)是计算机接收到中断信号到操作系统作出响应，并完成转换进入中断服务程序的时间。

与传统多任务操作系统类似，实时操作系统中的任务等同于分时操作系统中的进程的概念。系统中的任务有四种状态：运行、就绪、挂起、休眠：

- 运行:获得 CPU 控制权。
- 就绪:进入任务等待队列。通过调度转为运行状态。
- 挂起:任务发生阻塞，移出任务等待队列，等待系统实时事件的发生而唤醒。从而转为就绪或运行。
- 休眠:任务完成或错误等原因被清除。也可以认为是系统中不存在了。

系统中只能有一个任务在运行状态。各任务按级别通过时间片分别获得对 CPU 的访问权。

3.1.2 标准 Linux 在实时特性上存在的问题

实时系统的主要特性就是要求操作系统提供任务时限的保证，与时限的保证

性直接相关的是系统的最坏情况参数，同 Unix 一样，Linux 操作系统的设计目标是取得最优平均性能，因此很多方面无法满足实时系统的要求[24]：

◆ 进程调度问题

Linux 的内核任务是不可抢占的，采用基于固定时间片的可变优先级调度策略，当一个低优先级任务由于调度而进入核心状态后，除非当前进程需要等待资源释放而挂起，否则后来的高优先级进程只能等待当前进程完成系统调用，而系统调用的完成时间具有很大的不可预测性，这对一些要求高优先级进程立即抢占 CPU 资源的嵌入式应用是不能满足要求的。

◆ 关中断问题

在系统调用中，为了保护临界区资源，Linux 会长时间关掉中断，这样会加大中断延迟时间，阻塞高优先级的中断，使得它不能被立即被处理，在嵌入式实时应用中，这是一个十分严重的问题。

◆ 时间精度及定时器问题

操作系统必须对时间精度和时钟中断处理的时间开销进行考虑，时间精度越高，意味着时钟中断越频繁，而花在中断处理上的时间越多。Linux 通过对硬件时钟的编程产生周期为 100Hz 的时钟中断，因此任务调度的时间精度最高能达到 10ms，这无法满足一些对时间精度要求苛刻的嵌入式任务。

Linux 本身存在的这些问题，使得 Linux 无法支持实时任务，尤其是某些对时间有苛刻要求的任务。针对这种情况，人们提出了一些 Linux 内核的改造方案，下面具体分析两种具有代表性的实时化方案。

3.1.3 主流 Linux 内核实时化方案分析

对 Linux 进行实时化具有代表性的系统有两个：RTLinux 和 RTAI[25][26]

1、 RT-Linux 系统

RT-LINUX 系统可以说是所有实时 Linux 的鼻祖。目前最新版本已经发展到 3.0 版。RT-Linux 采用一个比较简单的做法，它根本不直接用 Linux 的任何功能，而把需要高度时间精确度的工作写成一个驱动程序的形式，然后直接用 PC 时钟芯片所产生的中断呼叫这个驱动程序。如此一来，不管 Linux 系统呼叫的时间有多长都不会影响系统的实时性能[27]。

2、 RTAI 系统

RTAI 是 Real-Time Application Interface 的缩写。顾名思义它是一套可以用来写实时应用程序的接口。RTAI 和 RT-Linux 在实现原理上基本相同。它同样的避开了对 Linux 内核的直接修改，而使用可装载内核模块当作实时进程。每一个实时进程实际上就是一个可装载内核模块。

3、 RTLinux 与 RTAI 的比较

RTAI 和 RT-Linux 最大的不同地方在于 RTAI 在 Linux 内核之上定义了一组实时硬件抽象层(Real-Time Hardware Abstraction Layer，以下简称 RTHAL)。RTHAL 将 RTAI 需要在 Linux 中修改的部份定义成一组程序接口，RTAI 只使用这组接口和 Linux 沟通。这样做的优点在于可以将直接修改 Linux 内核的程序代码减至最小，使得将 RTHAL 移植到新版 Linux 的工作量减至最低。

RTAI 采取这种机制的原因在于 RT-Linux 在由 2.0 版内核移植至 2.2 版内核的过程时遇到问题，使得基于 2.2 版内核的 RT-Linux 一直无法完成。促

使 RTAI 的开发者 Paolo Mantegazza 和他的同事开始自行做移植的工作，但由 RT-Linux 的困境他们决定采取上述的机制来解决将来可能再度面临的兼容性问题。

RTAI 就是在这种情况下出现的，可以说它是一个比 RT-Linux 更具有扩充性的 RT-Linux，虽然后来 RT-Linux 也随后完成移植的工作，但是已经落后 RTAI 半年了。

两者主要差异为：

- 支持平台： RTAI 支持 i386, MIPS, PPC, ARM, m68k-nommu，而 RTLinux 只支持 i386, PPC, ARM 平台。在嵌入式系统的移植性上，RTAI 要明显高出一筹。
- 开发进度： RTAI 目前仍在开发中，最新版本为 24.1.10。而 RTLinux 已经停止开发进程。

其余主要不同点见表 3-1：

表 3-1：RTLinux 与 RTAI 的比较

	RTLinux	**RTAI**
开发者	FSM Labs	DENX Software Engineering, Pengutronix and Sysgo Realtime Solutions
支持平台	i386, PPC, ARM	i386, MIPS, PPC, ARM, m68k-nommu
使用权	商业 GPL	LGPL - GPL
应用编程接口	POSIX	自定义，符合一定 POSIX 标准
存储	共享	动态、共享
调试	GDB, Event tracer	Cross GDB, LTT
进程间通讯	FIFO's	FIFO's, Named FIFO's, Mailboxes, messages, PQueues, net_rpc
用户态实时	Signal handlers	LXRT soft, LXRT hard, Mini LXRT
其他	RT-Lab	Octave, proc, Watchdog, Scripting, Xenomai

鉴于这种情况，本课题重点分析了 LEOS 实时化的实现原理与机制，设计 LEOS 实时化方案。

3.1.4 LEOS 双内核实时化解决方案

3.1.4.1 LEOS 实时化原理

由于 Linux 不是针对实时系统而设计的，所以 Linux 在处理实时任务时需要对 3.1.1 节中分析的问题进行改进。而直接修改 Linux 的内核并不是一个很好的方法，因为 Linux 内核是一个非常庞大的工程，直接修改工作量过大，容易引起系统的不稳定。同时 Linux 内核还在不断升级，直接修改将使实时化工作移植到新版本内核的难度提高，如果新版本内核改动很大，甚至可能导致移植的失败。

根据以上分析，LEOS 内核实时化基本策略是为 Linux 增加一个实时内核，将 Linux 原来的内核作为实时内核的一个非实时进程，而所有实时进程直接与实时

内核交互，并由其进行管理，这种方法使得实时进程得以尽快的被执行。基本结构如图 3-1 所示：

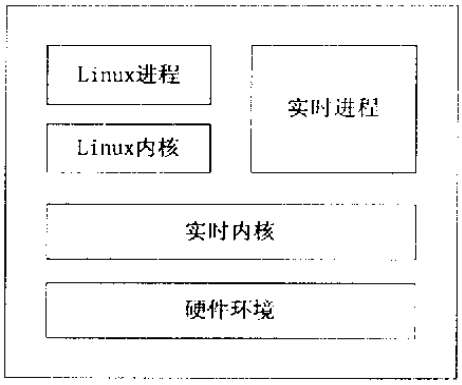

图 3-1 Linux 实时化基本结构图

图 3-1 中，在 Linux 进程和硬件中断之间，本来由 Linux 内核完全控制，现在在 Linux 内核和硬件中断的地方，插入了一个实时内核的控制。Linux 内核只是作为实时内核的一个非实时进程而存在，通过控制实时内核的进程调度策略可以改变 Linux 对实时进程的调度方法。解决了 3.1.2 节中的进程调度问题。

在实时化的 Linux 中，控制信号都要先交给实时内核先进行处理。在实时内核中实现了一个虚拟中断机制，Linux 本身永远不能屏蔽中断，它发出的中断屏蔽信号和打开中断信号都修改成向实时内核发送一个信号，设置实时内核中的某些标记位，也就是说将所有的中断，分成 Linux 中断和实时中断两类，如果实时内核收到的中断信号是普通 Linux 中断，那就设置一个标志位；如果是实时中断，就继续向硬件发出中断。因此 Linux 不能中断自己，而实时内核可以，解决了 3.1.2 节中的关中断问题。

对于实时系统时钟和中断的控制非常重要，可以单纯对时钟芯片进行编程，提高时钟频率，增加系统时间精度。但这样会极大消耗系统资源，因为在这种情况下，无论是否有事件发生，CPU 总是被频繁中断。而且实时任务并不会每一个时间片段都会发生，而是要求可以在任何时刻发生。实时内核在这方面通过提供另外一种时钟模式来解决，将时钟芯片设置为下一个时间发生的时间，这种设置方式就是一次模式。一次模式解决了 3.1.2 节中的时间精度和定时器问题。

3.1.4.2 LEOS 双内核实时化设计方案

根据嵌入式的特性，在设计时应考虑如下问题：

- 嵌入式系统中，处理器处理能力十分有限，而且嵌入式应用通常比较单一，所以在实时化过程中，应使得系统结构尽量简单，减少对系统资源的占用，提高系统效率。
- 嵌入式系统对软件所占用的存储空间也有较高要求，应尽量减少实时化所带来的系统空间膨胀。
- 由于 Linux 的内核功能非常强大，出于减少系统复杂性和增加系统稳定性考虑，对于实时内核，如果需要的是 Linux 内核中已有的功能，则不在实时内核中做具体实现，而是使用 Linux 内核已有的接口。

本文在参考 RTAI 的实现原理基础上，根据以上设计原则和上节分析的原理，设计了 LEOS 的实时化方案，其结构如图 3-2 所示。

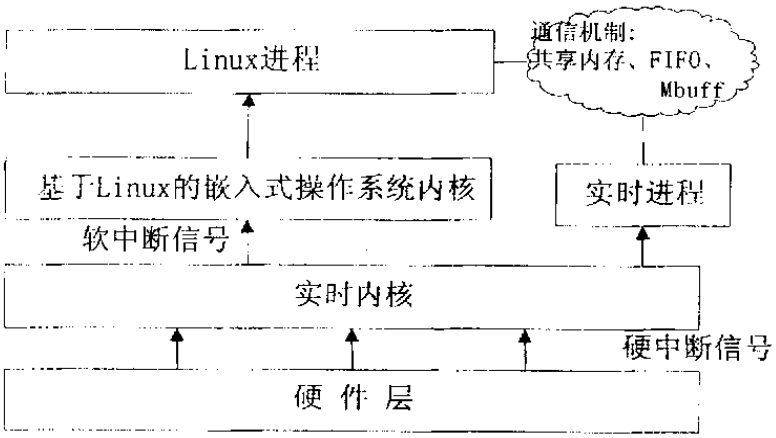

图 3-2 实时化的双内核结构图

实时化方案对 Linux 内核本身不进行大的修改，而是采用双内核机制，设计一个小而简单的实时内核进行实时任务管理，提供任务的可抢占式调度，并使用 Linux 本身作为这个实时内核的优先级最低的任务，所有的实时任务的优先级都要高于 Linux 本身以及 Linux 下的一般任务，这就保证了实时任务能够高于普通任务获得处理器。

出于对嵌入式资源的有效利用和系统的灵活性考虑，调度方法、用户实时任务通过 Linux 的可装载模块的方式进行的。当用户不需要进行实时任务处理时，卸载所有的实时模块即可以使得实时内核不进行实时处理，系统仍然按照标准 Linux 来处理任务，这种模式下，实时内核不会过多的占用嵌入式有限的资源；当用户需要进行实时任务处理时，再动态加入实时模块，或者用户也可以自定义实时调度算法，通过 Linux 的模块插入到内核运行空间，作为实时任务的调度策略，用户实时任务也通过模块编程来实现的，均可以动态加入或卸载。

使用可装载模块的机制，最大限度的减少了对系统资源的浪费。因为在实时模式下，为了能够达到实时标准所做的修改如中断转发等处理会浪费系统的一部分处理能力，而且会影响非实时任务的执行，如果在系统没有实时任务时，仍然保持系统的实时特性会降低系统的使用效率。所以使用模块化设计可以根据任务的需要随时调整系统的处理方式，不会将有限的资源浪费在无用的处理上。同时这种模式也保证了系统的可扩充性和灵活性。

LEOS 实时内核的对实时任务的其它主要解决方案如下：

1、对于中断处理，实时内核首先截获外围设备的中断信号，如果是非实时任务则直接转发到 Linux 原始内核，对于非实时任务实时内核完全是透明的；如果是实时任务，则实时内核直接进行处理，由于实时任务的优先级要高于 Linux 的所有任务，实时内核中的处理能够立即进行，解决了 Linux 中存在的中断处理时延问题。

实时内核的存在，使得 Linux 本身永远不能屏蔽中断，它发出的中断屏蔽和打开中断信号都修改成向实时内核发送一个信号。如在 Linux 里面使用“sti”和“cli”宏指令来屏蔽和使能中断，是通过向 x86 处理器发送一个指令，实时内核修改了这些宏指令，执行时只是在实时内核里面的某些标记做修改。也就是说对所有的中断，分成 Linux 中断和实时中断两类，如果实时内核收到的中断信号是普通 Linux 中断，那就设置一个标志位，等待调度；如果是实时中断，就真正的向硬件发出中断，解决了 Linux 下关中断问题，处理流程如图 3-3 所示。

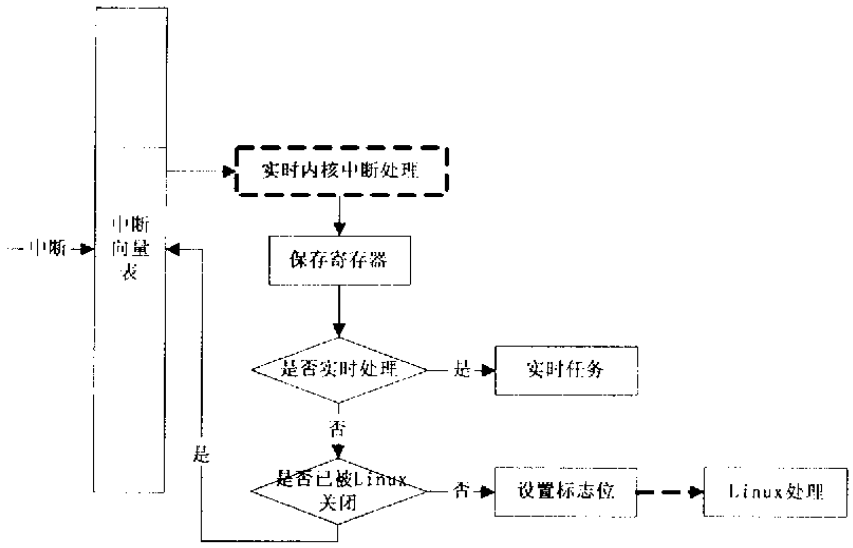

图 3-3 中断处理流程图

2、由于采用了双内核的结构，当用户同时使用实时和非实时任务时，就要求两个任务之间能够进行通讯和同步。由于 Linux 被处理为非实时任务，使得在实时内核的实时任务难以直接使用系统调用，它必须通过特定的方法和 Linux 进程进行通信。参考标准 Linux 的通讯方法，实时内核提供了三种通信机制：

- 共享内存：在实时内核启动的时候，通过指定给内核一个 mem 参数决定内核可以使用的内存大小，空出来的内存空间用于实时任务和 Linux 进程进行通信的共享内存。在实时内核任务中通过/dev/mem 设备在这段内存中寻址，Linux 进程也通过读取这段内存的数据获得实时任务提供的信息，完成实时任务和 Linux 进程之间的通信。
- FIFO 设备：利用一种特殊属性的管道进行通信。通过在/dev/下面创建 FIFO 的字符设备。实时任务可以往这个字符设备写数据，非实时任务可以从这个设备中读取数据。对 FIFO 设备的读写是不同步的，非实时任务对设备的读取并不会影响实时任务对它的写入。
- mbuff 驱动程序:这是一种使用共享内存的方法，实现内核内存空间和用户内存空间之间的共享。提供 mbuff_alloc（）函数对申请的内存取一个名字，mbuff 驱动程序使用一个链表通过这个名字来管理这些申请的内存。通过这个驱动程序可以在包括实时内核任务的 Linux 内核内存空间和用户内存空间之间共享内存。

3、实时内核在时钟处理上采用一次模式和周期模式结合的方法，为时钟芯片(8254)重新编写程序，使得系统时钟中断既可以随到随处理，也可以周期处理。提高了系统的响应能力，使得实时应用得到支持，又不会把嵌入式系统有限资源过多的浪费在频繁的中断上。引入这种方法后，系统可以根据不同的实时应用对任务响应时间的需要，选择合适的定时器和时间精度。

通过这样的设计就解决了 LEOS 实时性问题，由于采用双层内核结构和可动态装载的模块机制，很好的解决了嵌入式系统处理能力和实时任务之间的矛盾，同时使得设计方案也具有很好的可移植性。

3.2 LEOS 的文件系统技术研究

由于大多数嵌入式系统体积都很小，所以嵌入式系统一般不能象 PC 那样使用大体积的存储器，比如 IDE 硬盘等。很多应用中，嵌入式系统都使用闪存(Flash)做为它的永久存储系统，嵌入式系统需要针对 Flash 建立文件系统，Flash 存储器与磁存储器在工作原理上差别很大，一般的文件系统不能完全发挥 Flash 的特性，这就需要为 Flash 建立专用的文件系统。

3.2.1 Flash 简介

Flash 是一种电子可擦写只读存储器[28]，也叫做闪存，分为两种：一种是直接可操作的 NOR Flash，另一种是通过一个 8 位总线传输数据和地址的 NAND Flash，NAND 拥有单独的控制总线。

Flash 是大多数嵌入式设备用来存储操作系统的专用的存储器。它具有允许操作系统升级的优点，还可以用于数字式蜂窝电话、数字式照相机、LAN 交换机、PC 卡、数字式机顶盒、嵌入式控制器和其它小型设备。

所有的 Flash 芯片均被分割成很多“块”，通常 NOR Flash 使用 128KB 一块，而 NAND 则使用 8KB 一块。所有 Flash 的操作最小单位都是块，比如，当需要重写数据，即使只修改某个块中的一位数据，也必须将整个块重写。通常一块 Flash 可以被重写 100,000 次左右。由于 Flash 只能修改一定次数，所以为了保证某一块不会在其余块之前用完所有擦写次数，通常会使用轮流的方法使用每个 Flash 块，这种优化的方法被称为：Wear leveling[29]

目前对 Flash 常用操作方法是将其虚拟为一个块设备，然后在其上架设文件系统，比如将虚拟块设备一对一映射到 Flash 芯片上，这样当需要写入数据时，先将整个数据块读出，进行修改，然后擦去原来的数据，最后将新的数据整块写回 Flash。这种做法没有提供“Wear leveling”优化，并且非常不安全，如果在擦去旧数据和写入新数据之间，系统突然断电，就会造成文件的丢失，甚至是整个文件系统的瘫痪。尽管如此，这种方法在工业产品中仍然可以使用，因为很多产品的文件系统是只读的，不存在修改时出现错误的情况。

为了提供 Flash 文件系统的“Wear leveling”，就诞生了闪存转换层[29] (Flash translation layer,以下简称 FTL)，在 FTL 中，虚拟的文件块不会被放在同一个真正的 Flash 块上，而是采用轮换的算法，使用不同的数据块，这样就保证了即使文件系统对某一个块的操作比较频繁，但真正的 Flash 不会存在某一块比其余块多擦写很多次的情况，FTL 就是用来保存这个映射关系的中间层。

3.2.2 JFFS 文件系统

嵌入式应用中，通常使用电池做为系统的电源，由于电池的电量限制，难以持续为系统供电，系统很容易突然断电，文件系统如果在断电时正在进行操作，容易造成数据的丢失，在重新启动后，文件系统需要进行全面的扫描以进行系统的恢复工作，这个过程往往耗时过长，而且如果断电时正在进行关键数据操作，

文件系统甚至有可能因此而毁坏。这就要求嵌入式中所使用的文件系统必须具有具有更高的稳定性、容错性和可恢复性。日志闪存文件系统[30](Journaling Flash File System,以下简称 JFFS)是针对 Flash 设备的特性而设计的文件系统，它专门针对嵌入式中可能出现的断电恢复、Flash 的 Wear leveling 而设计，很好的解决了这些问题。

JFFS 采用的是日志文件系统机制，日志文件系统相对于普通文件系统，最主要的改变是增加了日志记录。在数据库管理系统中的日志技术已经相当成熟，而且有较长时间的实际应用，证明是非常高效、稳定的。日志文件系统的最重要一个原则就是，先写日志，再写数据，这样当系统发生突然性的断电等事故后，系统重启会自动根据日志记录把尚未完成的文件操作取消，以保证文件系统的一致性和完整性。下面讨论 JFFS 具体的实现机制。

3.2.2.1 JFFS 存储格式

与一般的日志文件系统采用额外的日志记录不同，JFFS 的数据操作完全按照日志方式保存在 Flash 设备上，所有操作都是按照日志来进行管理的。JFFS 采用了纯粹的线性存储方式，数据节点和元节点均按完全线性方式顺序写入 Flash 芯片，类似于将 Flash 芯片当作一个日志文件来组织文件系统。

在 JFFS1(JFFS 第一版)文件系统中，所有数据均存储在 jffs_raw_inode 节点中，每个 jffs_ram_inode 节点均属于某个索引节点（inode），每个节点的开始是它所属的索引节点号和所有该索引节点的元数据。每个 inode 下的所有节点都存储一个顺序的序列号，用来代表该节点的版本号，节点写入时都会选取一个比所属 inode 下所有节点版本号都高的版本号，以保证节点版本号的唯一性。版本号有 32 位，可以保证每个 inode 下存储高达 4 百万个节点，由于 Flash 的擦写寿命有限，所以可以保证在 Flash 的整个寿命过程中每个 inode 下的节点可以获得不同的版本号。同样每个 inode 的版本号也是采用 32 位的，每个 inode 的版本号都不会被重用，采用这样的方法就保证了文件系统恢复过程中的完整性和一致性。

相对通常文件系统的节点，JFFS 文件系统的节点除了存储 udi、gid、mtime、atime 等之外，还存储它所属的 inode 节点名和节点号。当节点中存储有数据时，节点还会记录该数据在文件中的偏移位置，所有这些增加的内容都是为了在需要的时候能够完整的恢复中断前的文件系统。

对于特殊的 inode，如字符设备、块设备或符号链接的 inode，要求其最多只拥有一个节点用于存储数据，如设备号、字符链接的字符串等。因为这些文件所需要的空间一般都很小，而且通常这些文件的数据都要求一次读取完整，所以要求这些文件只存储于一个块中，会使文件系统的操作简单不少。

删除 inode 时，需要在相应的 inode 节点元数据处设置删除位，并设置所有该 inode 下的节点删除位，当所有该文件的当前应用被关闭时，这些节点就会成为废弃节点。

在 JFFS2 也就是第二版的 JFFS 文件系统中，不再仅仅只有一种节点，在这一点上 JFFS 更有灵活性，它允许在升级时定义新的节点，每个节点头部如图 3-4 所示：

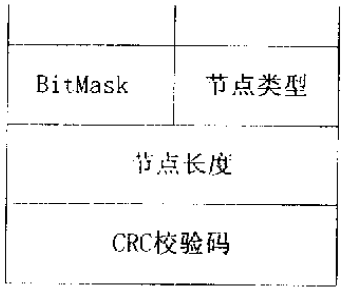

图 3-4 JFFS 文件节点头部结构图

3.2.2.2 JFFS 操作

当 JFFS 文件系统挂载时，系统会扫描整个 Flash 存储空间，由于每个节点均记录有所有上下级关系，所以可以依据每个节点的信息建立整个文件系统和所有 inode 的物理地址结构。JFFS 会在挂载文件系统时将所有数据存储起来，这样每次目录的读取可以在内核进行，然后直接定位到相应的 Flash 数据块，读入缓冲区。

当原数据被改变，如所有者或读取权限改变时，系统并不改写原有的数据块，而是直接在日志文件系统的最后添加一个新的节点，并废弃原来的节点。当数据块被改变时，操作也是同样的。

JFFS2 在挂载文件系统时的操作与第一版非常类似，但它并不在内存中保存所有节点的信息，它只保存那些在需要时不能立刻读取到的信息，比如它保存所有 inode 信息，在挂载时，JFFS2 会建立一个 hash 表，并在需要时查询这个 hash 表以获得 inode 的物理位置。

当 JFFS2 文件系统装载时，需要进行 4 个步骤的操作：

1、扫描 Flash 上所有节点，检查 CRC 校验码，以确认节点的可用性，在这个过程中会建立原始节点表和 inode 的 hash 表，所有 inode 都会被加入到这个 hash 表中，节点的版本号也会在这个时候被读入，以免后期过多的扫描整个文件系统。

2、根据前面扫描的结果，建立整个文件系统的映象。

3、第二遍扫描整个 Flash，检查没有任何链接的 inode 节点，并将其删除。每当一个目录 inode 被删除时，扫描就要重新开始，因为目录 inode 的删除可能会导致新的废弃 inode。

4、再次扫描整个文件系统，并删除所有临时信息，只留下 inode 的 hash 表。

3.2.2.3 JFFS 废弃节点回收

当 JFFS 追加新节点，如果发现 Flash 已经无空间可用时，系统就需要对以前废弃节点的空间进行回收。

废弃节点回收发生的情况有两种，一种是内核线程请求空间，另一种是用户进程请求空间，当这两种请求发生时，如果 Flash 空间不够，而且文件系统中存在空间可以回收时，才会进行废弃节点回收。如果文件系统中不存在空间可以回收，也就是说存储空间已满时，系统会返回 ENOSPC 错误，提示系统空间不够。

当进行废弃节点回收时，系统总是从文件系统的第一块开始，如果第一块就是废弃节点则回收这块节点，并分配给请求空间的进程。如果不是，则头节点必须被设置成废弃节点，其中的数据依次后移，直到废弃节点被找到。否则返回 ENOSPC 错误。

以上是对 JFFS 文件系统的具体分析，对于通常的嵌入式应用，JFFS 文件系统能够较好的满足其要求，并能够提供类似于硬盘存储器的特性，但由于 Flash 的速度限制，使得 JFFS 文件系统的性能受到影响，尤其是在某些频繁进行文件操作的嵌入式系统中，文件系统的性能是很大的瓶颈。针对这种情况，本文设计了混合式文件系统，以提高系统的效率。

3.2.3 混合式文件系统解决方案

使用 JFFS 文件系统能够很好的解决系统突然断电情况下的文件系统恢复问题，也能够充分发挥 Flash 的特性，延长 Flash 的使用寿命。但使用 JFFS 文件系统也存在一些问题需要改进：

- 由于 Flash 的擦写过程比较复杂，使得 JFFS 文件系统的操作耗时过长，根据 4.8.1 节的测试，JFFS 文件系统的操作费时是嵌入式系统内存的 30 倍，是 IDE 硬盘的 300 倍，文件系统的性能严重影响了系统的效率。
- 从 JFFS 操作过程可以看出，JFFS 在启动时需要完全扫描 Flash 设备三次，该操作过程虽然可以保证系统的完整性，但操作时间过长，具体时间请参看 4.8.1 节的测试结果。
- 在标准 Linux 中，使用/proc 和一些临时目录用于暂时存放一些系统配置和参数，在系统关闭后，这些内容不需要保存，这些内容对于通常的用户没有直接使用的意义，尤其是嵌入式应用中，一般功能都比较简单，不允许用户对操作系统的配置和参数进行修改，所以这部分内容不需要保存在 Flash 上，浪费存储空间。
- Linux 有完善的日志功能，但在嵌入式应用中，这些功能并不重要，而且日志的记录会频繁操作文件，降低 Flash 使用寿命和系统性能。

这些问题的根本在于 Flash 的存储速度非常缓慢，在嵌入式应用中，内存的速度要比 Flash 快几个数量级，如果能够使用内存作为存储器可以极大提高系统性能。但内存的缺点就是无法保存数据，所以如果企图使用内存作为文件系统，则只能对一部分不允许用户修改的内容使用内存作为它在操作系统启动后的存储位置。

基于以上考虑，在实现中本文参考了 Linux 的 Ramdisk 机制，Ramdisk 机制允许系统使用一部分内存虚拟为文件系统，将这部分内存完全当作普通块存储器使用，并且对于操作系统，这是一个透明的模拟，所有操作都完全相同。

为了改进以上 JFFS 存在的问题，本文设计了混合式文件系统，其基本结构如图 3-5 所示。

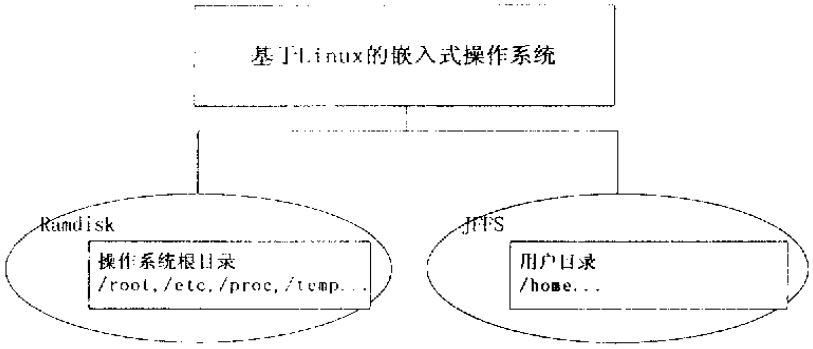

图 3-5 混合式文件系统结构图

基于对内存和 Flash 介质存储速度的综合考虑，混合式文件系统将系统中不允许用户更改和系统的在运行时临时建立的内容，如目录 root、proc、etc、temp 等，使用 Ramdisk 存储方式，直接放入 Ramdisk 映像中，采用压缩的方法存储在 Flash 上，系统启动时通过指定的 Flash 设备号，内核自动寻找其位置，经过解压缩后，直接放入内存，完成后，内核可以使用该部分内存做为文件系统，虽然内存在断电后会丢失所有修改，但存放在内存中的都是不需要修改的系统文件和无需保存的临时文件，断电对于文件系统没有影响。下次启动时，Flash 上的 Ramdisk 压缩映像仍然可以继续使用。

允许用户修改的部分，如目录 home 等，仍然采用 JFFS 方式，存储于 Flash 之上，在系统启动并装载 Ramdisk 内容后，通过启动参数指定的 Flash 设备号装载 JFFS 内容，由于 JFFS 可以永久记录所有操作，用户修改的数据就可以在 JFFS 中被永久保存。

嵌入式应用中，采用混合式文件系统，在为用户提供完整的功能基础上，既可以提高系统启动速度和文件操作性能，同时也减少了 Flash 成本。

3.3 LEOS 内核中文化技术研究

对于中文用户，系统能否正确显示中文至关重要，由于 Linux 开发者在设计时没有考虑国际化的问题，内核每次显示时都是以单字节的方式将内容显示在终端上，但中文是以双字节的存储的，如果不改动内核，那么中文在单字节模式下显示出来的将都是乱码，所以需要修改 Linux 中负责显示的底层处理，也就是本节所需要讨论的 Linux 的内核中文化问题。

3.3.1 Linux 字符显示原理

在讨论内核中文化方案之前，需要分析原有 Linux 的显示工作机制。这里主要涉及到两部分：Linux 下控制台和帧缓冲的实现[31]。

3.3.1.1 控制台（console）

通常在 Linux 下看到的控制台是由几个设备构成的。分别是/dev/ttyN（其中 tty0 就是/dev/console，tty1、tty2 就是不同的虚拟终端(virtual console)）。使用热键 Alt+Fn 在这些虚拟终端之间进行切换。这些 tty 设备对应于 linux/drivers/char/console.c 和 vt.c。其中 console.c 负责绘制屏幕上的字符，vt.c 负责管理不同的虚拟终端，并且负责提供 console.c 需要绘制的内容。

vt.c 把不同虚拟终端下的需要交给 console.c 绘制的内容，放到不同的缓存中去。vt.c 管理着这样一个缓冲区的数组，并且负责在这些缓存之间切换，并指定哪一个缓冲区是被激活的。用户所看到的虚拟终端对应着被激活的缓冲区。console.c 同时也负责接收终端的输入，然后把接收到的输入的信息放到缓冲区。

3.3.1.2 帧缓冲(FrameBuffer)

Framebuffer 的原理是将显存抽象成一个设备，可以通过对这个设备的读写直接对显存进行操作。这种操作是抽象的、统一的。用户不必关心物理显存的位置、换页机制等等具体细节，这些都由 Framebuffer 设备驱动程序来完成的。

Framebuffer 对应的源文件在 linux/drivers/video/目录下。总的抽象设备文作为 fbcon.c，在这个目录下还有与各种显卡驱动程序相关的源文件。 在使用帧缓冲时，Linux 将显卡置于图形模式下。

以一个简单的例子来说明字符显示的过程。假设是在虚拟终端 1(/dev/tty1)下运行如下的简单程序：

```
main ()
{
    puts(" hello, world.\n" );
}
```

pputs 函数向缺省输出文件（/dev/tty）发出“写”的系统调用 write(2)。系统调用到 Linux 内核对应的内核函数->——console.c 中的 con_write( )，con_write()最终会调用 do_con_write()，在 do_con_write()中负责把"hello, world.\n"这个字符串放到 tty1 对应的缓冲区中去。 Do_con_write()还负责处理控制字符和光标的位置，do_con-write()这个函数的声明：

```
Static int do_con_write (struct tty_struct * tty,
                         int from_user,
                         const unsigned char *buf,
                         int count)
```

其中 tty 是指向 tty_struct 结构的指针，这个结构里存放着关于这个 tty 的所有信息（请参照 linux/include/linux/tty.h)。tty_srtuct 结构中定义了通用（或高层）tty 的属性（例如宽度和高度等)。

在 do_con_write ( ) 函数中用到了 tty_struct 结构中的 driver_data 变量。Driver_data 是一个 vt_stuct 指针。在 vt_struct 结构中包含这个 tty 的序列号（正使用 tty1，所以这个序号为 1)。Vt_struct 结构中有一个 vc 结构的数组 vc_cons，这个数组就是各虚拟终端的私有数据。

```
Static int do_con_write(struct tty_struct * tty,
                        int From_user,
                        const unsigned char *buf,
                        int conut)
{
```

```
    struct vt_struct *vt = (struct vt_struct *)tty->driver_data;
    /用到了 driver_data 变量
    ..........
    currcons = vt->_num;
    //在这里的 vc_nums 就是 1
    ..........
}
```

要访问虚拟终端的私有数据，需使用 vc_cons[currcons].d 指针。这个指针指向的结构含有当前虚拟终端上光标的位置，缓冲区的起始地址、缓冲区大小等信息。

“hello,world.\n”中的每一个字符都要经过 conv_uni_to_pc()这个函数转换成 8 位的显示字符。这样做的主要目的是使不同语言的国家能把 16 位的 Unicode 码映射到 8 位的显示字符集里，目前主要还是针对欧洲国家的语言，映射结果为 8 位，不包含双字节(double byte)的范围。

经过 conv_uni_to_pc()转换之后，” hello, world.\n” 中的字符被一个一个地填写到 tty 的缓冲区中，然后 do_con_write()调用底层的驱动程序，把缓冲区中的内容输出到显示器上（也就相当于把缓冲区的内容拷贝到 VGA 显存中去）

```
    sw->con putcs(vc_cons[currcons].d,
    (u16 *)draw_from, (u16 *)draw_to_
    (u16 *)draw_rwom, Y, draw_x);
```

之所以要调用底层驱动程序，是因为存在不同的显示设备，其对应 VGA 显存的存取方式也不一样。

上面的 Sw->con_putcs() 就会调用 fbcon.c 中的 fbcon_putcs()函数(con_putcs 是一个函数的指针，在 Framebuffer 模式)下指向 fbcon_putcs()函数，也就是说，在 do_con_write()函数中是直接调用了 fbcon_putcs()函数来进行字符的绘制，比如说在 256 色模式下，真正负责输出的函数是：

```
static void fbcon_putc(struct vc_data *conp,
                       int c,
                       int ypos,
                       int xpos)
```

在 fbcon_putc 中真正负责输出的函数是底层函数：

```
void fbcon_cfb8_putcs(struct vc_data *conp,struct display *p,
                      const unsignde short *s,
                      int count,
                      int YY,
                      int xx )
```

3.3.2 内核中文化解决方案

在分析 Linux 下字符显示的原理和显示过程的基础上，本文提出了针对嵌入式操作系统的内核中文化方案。

1、根据分析，Linux 内核中的中文化主要方法是截断 Linux 的字符分析处理，加入是否为中文字符的判断，如果是中文字符则进行与其他字符不同的显示映射，然后根据映射表读取字库，并将显示内容放入帧缓冲中，显示字符。

2、具体实现时，首先为 Linux 加入从 Unicode 到中文字符的映射表，该映射表能够不改变英文和控制字符的映射，而将双字节的中文映射到相应的字符上。

然后改进 Linux 的字符显示过程，加入针对中文显示的处理，数据处理流程如图 3-6 所示。

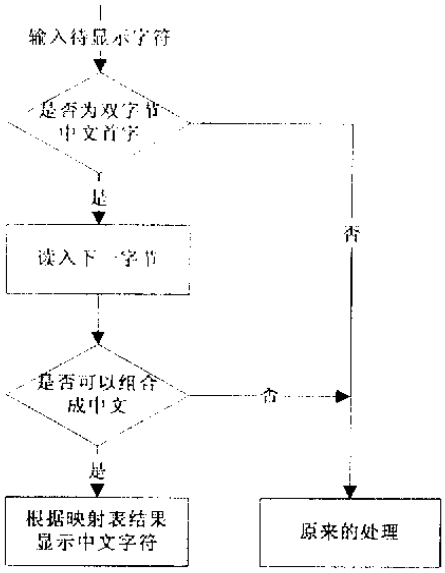

图 3-6 内核中文化数据处理流程图

3、在内核处理中文字符显示的算法如下：

算法 1：

1) 输入：待显示字符 c
2) 输出：待显示帧缓存
3) **if** UniconFontManager == NULL **then**　　\\判断控制台是否已初始化为中文
4) 　　UniconFontManagerOpen ();　　\\如果不是，则初始化控制台为中文
5) **end if**
6) **if** IsChinese(c) **then**　　\\判断 c 是否为中文字符首字
7) 　　c2=GetNextChar(c);　　\\如果是，读取下一个字符，组成完整的中文字符
8) 　　GetUni();　　\\获得字符映射
9) 　　Getbuffer();　　\\根据汉字字库获得相应的显示内容
10) 　　Putbuffer();　　\\将汉字显示内容写入帧缓冲，交给底层驱动显示
11) **else**
12) 　　DoNormal();　　\\如果不是，则进行正常处理
13) **endif**

3.4 小结

本章讨论并解决了将 Linux 改造为嵌入式操作系统所涉及到的主要技术难点，通过考察目前流行的技术趋势，分析了解决这些问题所需要考虑的内容和所涉及的技术原理，在此基础上提出了双内核的实时化方案、混合式文件系统方案和内核中文化方案。下章将介绍 LEOS 在 StrongARM 硬件平台上的设计与实现。

第四章 StrongARM 平台上 LEOS 的设计与实现

本文选择 Intel 公司的 StrongARM 硬件平台做为 LEOS 的实现基础，目标是为嵌入式应用开发通用的嵌入式操作系统，提供实时、中文等功能，并为平台上的所有硬件接口包括 Flash、显示屏、触摸屏、背光等设备提供支持。最终系统应能够满足通常的嵌入式应用需求。

本章首先概述了 StrongARM 开发环境，设计了 StrongARM 平台上的 LEOS 总体结构，并根据第三章中所讨论的关键技术分别介绍了它们在 StrongARM 平台上的实现，然后针对 StrongARM 平台的特性，进行与硬件平台相关的内核改造和启动过程优化，最后对 LEOS 性能进行了测试和评估。

4.1 StrongARM 开发环境

由于嵌入式平台本身无法搭建编译环境，所以进行嵌入式开发时需要在开发主机（通常是 PC）上建立一个对应于嵌入式平台的开发环境，用来进行嵌入式开发、编译和调试。

开发环境的建立主要是在开发主机上建立相应的工具链，创建一个用于编译将在 StrongARM 平台上运行的内核和应用程序的环境，这样做的原因是 StrongARM 平台的二进制执行级别与 PC 机不相兼容。

工具链[32]由一套用于编译、汇编和链接内核及应用程序的组件组成。这些组件包括：

Binutils — 用于操作二进制文件的实用程序集合。它们包括诸如 ar、as、objdump、objcopy 这样的实用程序。

Gcc — GNU C 编译器。

Glibc — 所有用户应用程序都将链接到的 C 库。避免使用任何 C 库函数的内核和其它应用程序可以在没有该库的情况下进行编译。

构建工具链的目的是为建立一个交叉编译环境。其中交叉编译器的功能是运行在某一种处理器上，而目标是编译另一种处理器的指令。本开发系统中在 PC 开发机上编译目标 StrongARM 平台上的可执行指令。设置交叉编译器工具链包括：下载源代码、修补补丁、配置、编译、设置头文件、安装以及其他辅助的工作。本系统使用的是 Intel 提供的开发工具链[33]，在主机上建立开发环境的具体步骤如下：

1、建立 StrongARM 平台内核的头文件和链接
2、建立实用程序集合
3、建立 StrongARM 平台 C 编译器
4、建立 StrongARM 平台 Glibc 库
5、建立 StrongARM 平台 C++编译器
6、进行应用程序的开发与测试

在主机上建立开发环境后，将 StrongARM 开发板通过并口与主机相连，使用 Jflash2 工具向 StrongARM 开发板的 Flash 上写入引导程序，然后按照图 4-1 连接开发板与主机：

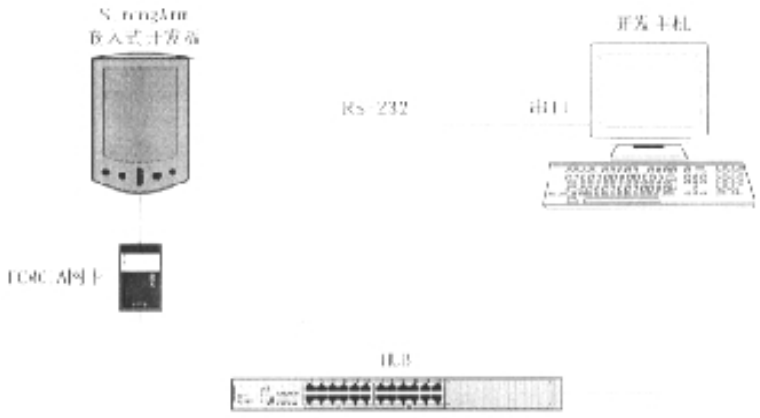

图 4-1 开发环境结构图

开发主机与嵌入式开发板通过串口相连接，主机可以通过串口与引导程序通信，控制开发过程。开发板通过 PCMCIA 网卡接入局域网，以进行内核和文件系统的下载，并可以测试嵌入式系统的网络功能。开发主机上通过串口通信工具 minicom 与开发板上引导程序通信，获得开发板的信息，如开发板的 Flash 设备结构可以通过在引导程序下执行 fis list 命令获得：

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
RedBoot           0x50000000  0x50000000  0x00040000  0x00000000
RedBoot config    0x51F80000  0x51F80000  0x00001000  0x00000000
FIS directory     0x51FC0000  0x51FC0000  0x00040000  0x00000000
zImage-ram        0x50040000  0x00100000  0x00100000  0x00000000
ramdisk           0x50980000  0x00800000  0x00300000  0x00000000
zImage-jf         0x50380000  0x00100000  0x00100000  0x00000000
jffs2new          0x50C80000  0x00800000  0x00F00000  0x00000000
```

至此，开发环境已经搭建完成。基于已搭建完成的开发环境可以进行操作系统的开发和研究，由于嵌入式开发会涉及到两个平台，而且是本文是操作系统层次的实现，完全不同于通常的软件开发，通常的开发基本步骤如下：

1、在开发主机上建立对应平台，也就是 StrongARM 的工作链，设置交叉编译器、开发工具、内核等。

2、使用开发主机并口为 StongaArm 开发板写入引导程序做为系统启动和管理 Flash 的 BootLoader。

3、在开发主机上建立 StrongARM 平台的内核，交叉编译新内核，使用引导程序上传至开发板。

4、在开发主机上为开发板建立文 JFFS 文件系统映像或 Ramdisk 文件系统映像，使用引导程序上传至开发板。

5、在引导程序中配置系统启动参数，指定内核和文件系统。

6、启动系统。

通过上述步骤可以完成搭建整个 LEOS，对第三章的设计内容进行开发，完成后的系统目前已在其上进行一些应用的开发，基于 MiniGUI 图形界面的车辆状况模拟监视系统的屏幕效果如图 4-2 所示：

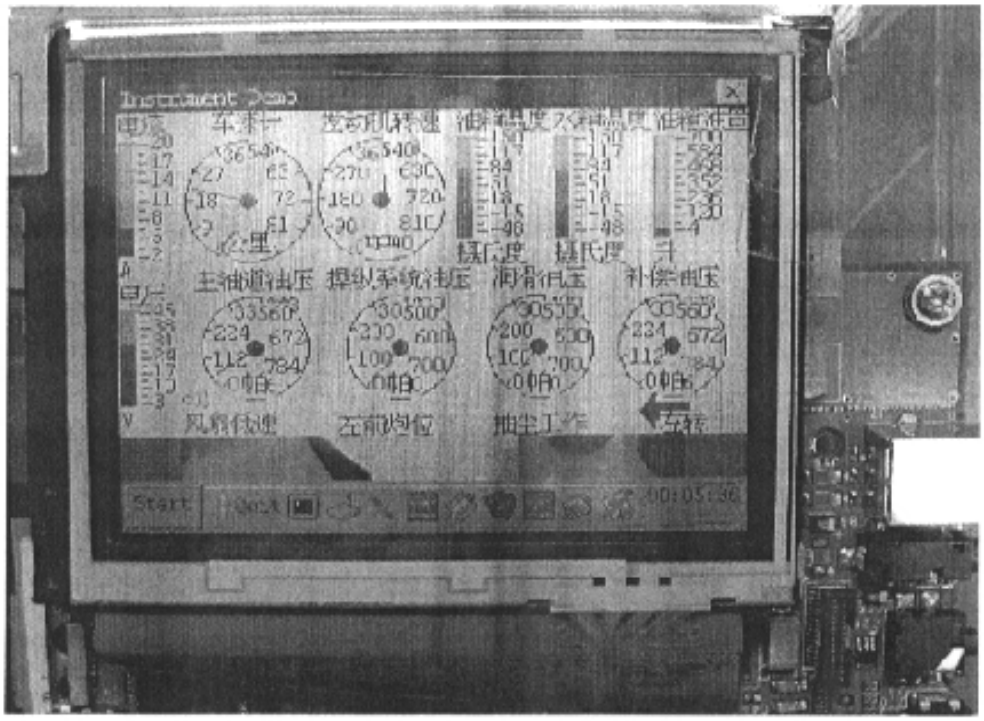

图 4-2 LEOS 上的应用程序界面图

4.2 StrongARM 平台上的 LEOS 整体设计

根据上一章针对关键技术的研究，本文设计了 LEOS 在 StrongARM 平台上的系统整体结构，如图 4-3 所示

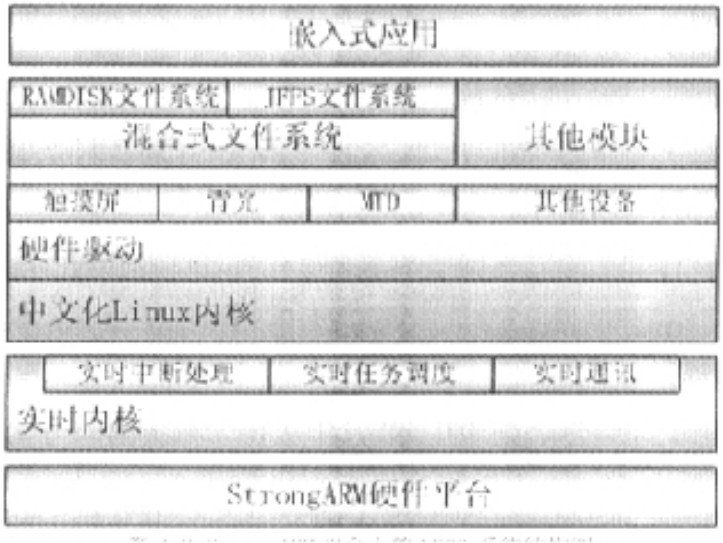

图 4-3 StrongARM 平台上的 LEOS 系统结构图

系统的实现可以分为两部分：平台无关和平台相关。其中平台无关部分包括：实时内核、文件系统和中文化内核部分，平台相关部分主要是针对 StrongARM 平台进行的内核改造，包括硬件驱动的改造和启动过程改造。

由于平台无关部分会影响到平台相关部分的实现，本章中首先讨论了平台无关部分的设计与实现，包括 LEOS 的实时内核、文件系统和内核中文化。然后针对 StrongARM 平台对 LEOS 的 Linux 内核进行相应的改造。

4.3 LEOS 的实时内核

根据设计中的实时化方案，本系统实时化实现中采用模块形式，当需要实时支持时，可以动态向内核中插入模块，建立实时内核。其基本支持模块包括：

1、Rt:这是实时的核心模块，所有的服务均需要这个模块的支持，该模块初始化所有的控制变量和数据结构，复制 Linux 的 idt_table 和中断向量地址，并初始化中断芯片的控制函数。

2、Rt_sched: Rt_sched 模块负责对任务进行调度，包括 Linux 内核。当发生系统调用或者时间中断时，该模块将选择处于 Ready 状态的优先权最高的任务，分配给它 CPU 使用权。

3、Rt_fifos: Rt_fifos 模块为实时内核实现了先来先服务 (First in First Out，简称 FIFO)功能。因为很多应用同时使用实时任务和普通 Linux 任务，如日志管理和显示系统，管理部分使用实时任务，而显示则使用普通任务，两者间需要一个 FIFO 缓存进行通讯，实时端的 FIFO 操作可以完全使用 Rt_fifos 模块提供的函数调用，而普通任务可以将 Rt_fifos 当作一个字符设备来进行操作。

4、Rt_shm: Rt_shm 为实时和普通任务提供了共享内存的服务，这个服务是对等的，比如，同样的调用，实时任务和普通任务都可以使用。第一次内存请求会真正的申请一段空间，而任何接下来的同名请求均会返回相应的地址空间，同样释放空间的过程也是一样的，每次释放并不真正的回收空间，直到最后一次释放才进行真正的操作。

5、Lxrt: Lxrt 模块为实时内核的调度功能提供了必要的服务，比如各种任务间的通讯功能：Linux<->Linux, Linux<->实时内核 和实时内核<->实时内核等各任务间的共享内存、消息发送和时间功能等。

6、Rt_pqueue, Rt_pthread, Rt_utils: 这些都是 Posix 标准的模块。Rt_pthread 提供了实时环境下的线程支持，Rt_pqueue 提供了内核模式下安全的消息队列服务。

在动态装载这些模块后，内核具备了处理实时任务的能力，并可以通过不同的任务调度模块改变系统的任务调度策略。也可以通过编制实时模块的方法，处理实时任务。

4.4 LEOS 的文件系统

根据上一章的设计方案，混合式文件系统，采用 RAMDISK 文件系统和 JFFS2 文件系统混合使用的方式，RAMDISK 文件系统将内存中的某一块模拟为硬盘空间，从而可以象对待硬盘空间一样在其上保存文件，由于内存的存取速度比一般的硬盘或者 Flash 介质要快很多，所以 RAMDISK 可以极大提高系统的性能，虽然内存中的数据在断电以后会全部丢失，限制了 RAMDISK 不能用于永久保存数据。

对于嵌入式系统，通常 Linux 内核及其相关的应用程序，并不允许用户随意修改，一般都是针对特定应用固化在系统中，所以虽然 Ramdisk 由于其不能保存修改难以能满足 PC 应用的需要，却可以很好的支持嵌入式应用。在实现中，将不允许用户修改的应用程序和 Linux 内核和应用程序等放入 Ramdisk 文件系统映像，并使用 Ramdisk 做为根文件系统，其余允许用户修改的部分放入 JFFS 文件

系统映像，在启动 Linux 操作系统后，将其装载入某个目录下，退出系统时再卸载之。

采用混合式文件系统时开发板的 Flash 使用状况如下表：

表 4-2 混合式文件系统 Flash 使用列表

名称	Flash 起始地址	内存起始地址	长度	偏移指针
RedBoot	0x50000000	0x50000000	0x00040000	0x00000000
RedBoot config	0x51F80000	0x51F80000	0x00001000	0x00000000
FIS directory	0x51FC0000	0x51FC0000	0x00040000	0x00000000
ZImage-ram	0x50040000	0x00100000	0x00100000	0x00000000
ramdisk	0x50980000	0x00800000	0x00300000	0x00000000
ZImage-jf	0x50380000	0x00100000	0x00100000	0x00000000
Jffs2new	0x50C80000	0x00800000	0x00F00000	0x00000000

该 Flash 中存有两个 Linux 内核：ZImage-ram 和 Zimage-jf，其中 ZImage-ram 是用以启动混合式文件系统的内核，文件系统也存储了两个：ramdisk 映像和 Jffs2new 映像，ramdisk 映像存储的是根文件系统，Jffs2new 映像存储的是用户可以修改的文件系统部分。

使用混合式文件系统时，设置 BootLoader 在启动时自动装入在 Flash 地址 0x50040000 的内核到内存空间 0x100000，并设置内核的启动参数为从 Ramdisk 启动，然后转向内存空间 0x100000 开始执行，启动内核，内核启动后会在内存中发现 Ramdisk 的文件系统映像，自动解开该映像并装载为根文件系统，运行其上的所有启动程序和服务，配置网络参数，在根文件系统装载后，设置启动程序装载 Flash 上的 JFFS 文件系统映像，并将其映射为 Ramdisk 根目录下的子目录/home，然后显示登陆界面，完成整个启动过程。

启动完成后，用户对/home 目录下的所有操作会被自动保存到 Flash 上，断电重启后仍然有效。而用户对其他目录的修改，因为是使用的 Ramdisk 文件系统映像，所以只在本次启动有效，断电重启后，所有修改都被放弃。

从以上实现可以看出，Ramdisk 文件系统映像适合用于保存内核、驱动、配置等不允许用户修改的内容，而 JFFS 用于保存用户的自定义数据。

4.5 LEOS 的内核中文化

在内核中加入了对中文显示的支持，由于中文字库大约需要 200k 的存储空间，对嵌入式系统而言这个要求可能并不能满足，实现中，LEOS 内核设计为可以根据需要选择是否加入中文字库，增加了内核的灵活性。主要实现方案如下：

根据上一章的分析讨论，原有内核的从 unicode 到显示字符的映射表，会把中文的字符映射到其他的字符上，造成显示错误。内核的中文化首先需要为内核建立新的映射表，加载符合双字节处理的映射关系，并且对控制字符进行一对一的不变映射，定制的符合这种映射关系的 unicode 码表是 direct.uni。

此外需要修改内核的打印函数，因为汉字是双字节的所以显示时需要考虑当前字节是否可以与下一个字节组成一个汉字，所以需要修改 driver/video/fb_con.c 源代码文件，加入函数：

```
void fbcon_putc_tl(struct vc_data *conp,
```

```
int c,
int ypos,
int xpos)
```

替代原有的 fbcon_putc，在这个函数中，首先建立 Unicon 的字符映射表，并判断控制台是否为双字节模式，若不是，仍然执行原来的处理。否则检查当前字符和后面的字符是否可以组成一个汉字，如果可以则设置 fbcon_cfb8_putcs 函数的 display 参数中的字体为双字节字体，并调用之。不可以则调用原来的处理。

其余改造工作主要是：

driver/char 目录下的：

console.c：增加支持中文的显示功能

direct.uni：新定义的从双字节到显示字符的映射表

fb_doublebyte.h：定义了支持中文所需要的函数和常量。

pc_keyb.c：增加了控制字符在中文支持下的处理

driver/video 目录下的：

Config.in：在内核配置的 Console drivers→Frame-buffer support 选项中增加“UNICON 支持”选项，用户可定制内核是否需要 UNICON（中文）的支持。

font_gb16.h：本文件为增加的 C 头文件，主要定义了基本汉字的字库。

4.6 LEOS 针对 StrongARM 平台特性的内核改造

由于 StrongARM 硬件平台的优越性能和广泛的使用范围，本文选取它作为实现基础。StrongARM 平台通常使用电池供电，而电池的电量是十分有限的，嵌入式设备对于电量损耗非常关注。同时由于嵌入式对存储容量的限制，使得在设计外部设备驱动时应尽量考虑做为系统的可选择模块，以节省对电量的消耗，并方便系统裁减。

根据目前 Linux 内核的发展、稳定性、对 StrongARM 开发板的支持以及对实时方案 RTAI 的兼容性考虑，最终选择了 2.4.17 版本的内核，该版本内核能够很好的符合上述要求，但需要针对 StrongARM 平台进行硬件驱动的改造。

4.6.1 触摸屏设备驱动

触摸屏设备在内核的 2.4.0-test9 版本下可以完全正常运行，并能够捕捉到用户输入。而 2.4.17 版本的内核中，触摸屏设备不能正常运行。所以采取将 2.4.0-test9 版本中的设备驱动移植到 2.4.17 版本中，主要改动在于 2.4.17 与 2.4.0-test9 版本内核中很多常数和宏的的定义不同，具体实现时，在 2.4.17 内核加入相应的定义，并为其建立相应的设备文件。

4.6.2 背光设备驱动

背光驱动的情况类似于触摸屏设备，考虑到背光发热量太大，长时间使用可能会损坏触摸屏和显示屏，而且根据统计，背光设备在嵌入式系统中，占到耗电量的 1/3 左右，非常容易导致系统电量用尽。所以 LEOS 中采用模块形式提供该

设备的驱动，系统启动后自动装载该模块，启动背光，当不需要时可以卸载该模块，关闭背光，以减少系统发热量和耗电量。在某些应用中，如手机、PDA 等，可以根据用户是否有输入动态启动/关闭背光设备，减少系统能耗。

4.6.3 闪存(Flash)驱动

MTD 设备是如 Flash 芯片、小型闪存卡、记忆棒等之类的存储器，它们在嵌入式设备中的使用正在不断增长，StrongARM 开发平台上就使用了 Flash 设备做为永久存储器。

MTD 驱动程序是在 Linux 下专门为嵌入式环境开发的新的一类驱动程序。相对于常规块设备驱动程序，使用 MTD 驱动程序的主要优点在于 MTD 驱动程序是专门为基于闪存的设备所设计的，所以它们通常有更好的支持、更好的管理并且基于扇区的擦除和读写操作的更好的接口。Linux 下的 MTD 驱动程序接口被划分为两类模块：用户模块和硬件模块。

- 用户模块：这些模块提供从用户空间直接使用的接口：原始字符访问、原始块访问、FTL 和 JFFS。
- 硬件模块：这些模块提供对内存设备的物理访问，但并不直接使用它们。通过上述的用户模块来访问它们。这些模块提供了在闪存上读、擦除和写操作的实际例程。

为了访问特定的闪存设备并将文件系统置于其上，需要将 MTD 子系统编译到内核中。这包括选择适当的 MTD 硬件和用户模块

有两个流行的用户模块可启用对闪存的访问：MTD_CHAR 和 MTD_BLOCK：

MTD_CHAR 提供对闪存的原始字符访问，而 MTD_BLOCK 将闪存设计为可以在上面创建文件系统的常规块设备（类似于 IDE 硬盘）。与 MTD_CHAR 关联的设备是 /dev/mtd0、mtd1、mtd2 等，而与 MTD_BLOCK 关联的设备是 /dev/mtdblock0、mtdblock1 等，JFFS 文件系统就是建立在 MTD_BLOCK 设备之上的。

4.7 LEOS 针对 StrongARM 平台的启动优化

StrongARM 平台具有与 PC 完全不通的结构，SA1110 处理器在启动时需要进行初始化，某些 Linux 启动步骤在 StrongARM 平台上也可以省略或者优化。本节首先首先分析 BootLoader 的启动过程，然后根据 Linux 的启动过程，提出优化方法，最后讨论如何改进 LEOS 的启动界面。

4.7.1 Boot Loader 启动过程

StrongARM 平台的 SA1110 微处理器通电以后自动从内存地址 0 开始执行，也就是开发板中的 SDRAM 的最底端地址。由于这种特性，需要在该地址处存放一段程序，以进行系统的基本初始化，如硬件检测、与开发环境的通讯以及加载内核映像等，类似于 PC 上 BIOS 的工作。这段程序称之为 Boot Loader，在本系统中采用的是 Red Hat 公司开发的开放源码工具：Red Hat Embedded Debug and

Bootstrap[34] (RedBoot)，该工具能够为 StrongARM 平台提供引导功能，并且能够支持网络通讯、调试和简单的 FLASH 文件系统。

StrongARM 处理器允许操作系统改变内核时钟，以方便内存对于时间的处理。所以 Redboot 的第一个任务就是配置处理器时钟，并进行内存存取设置，一旦内存可以使用，就开始初始化相应的堆栈。

内存初始化完成后，Redboot 的任务就是使能处理器上的串口中断处理，以允许开发板通过串口与开发环境通讯，此后 Redboot 还必须完成三个基本工作：

1、禁用 Memory Management Unit(MMU)

2、寄存器 r0 置零

3、寄存器 R1 置相应的体系结构代码，SA1110 为 0x19

完成这些后，Redboot 的工作就已经完成，可以通过用户命令或者系统自动执行进入下一步，Redboot 将跳至内存中内核代码的开始处，开始启动内核。

4.7.2 内核启动过程

内核第一步开始检测系统体系结构，也就是寄存器 r1 中存储的数据，完成以后内核开始建立页表，为 BSS 寄存器清零，初始化栈指针，至此启动过程转入体系结构无关的通用 C 代码中。

下一步内核开始分配和初始化各种资源[35]

1、输出 Linux 版本信息（printk(linux_banner)）

2、设置与体系结构相关的环境（setup_arch()）

3、页表结构初始化（paging_init()）

4、使用"arch/arm/kernel/entry-armv.S"中的入口点设置系统自陷入口（__trap_init）

5、内核进程调度器初始化（包括初始化几个缺省的 Bottom-half，sched_init()）

6、时间、定时器初始化（包括读取 CMOS 时钟、估测主频、初始化定时器中断等，time_init()）

7、提取并分析内核启动参数（从环境变量中读取参数，设置相应标志位等待处理，(parse_options()）

8、控制台初始化（为输出信息而先于 PCI 初始化，console_init()）

9、剖析器数据结构初始化（prof_buffer 和 prof_len 变量）

10、内核 Cache 初始化（描述 Cache 信息的 Cache，kmem_cache_init()）

11、延迟校准（获得时钟 jiffies 与 CPU 主频 ticks 的延迟，calibrate_delay()）

12、内存初始化（设置内存上下界和页表项初始值，mem_init()）

13、创建和设置内部及通用 cache（"slab_cache"，kmem_cache_sizes_init()）

14、创建 uid taskcount SLAB cache（"uid_cache"，uidcache_init()）

15、创建文件 cache（"files_cache"，filescache_init()）

16、创建目录 cache（"dentry_cache"，dcache_init()）

17、创建与虚存相关的 cache（"vm_area_struct"，"mm_struct"，vma_init()）

18、块设备读写缓冲区初始化（同时创建"buffer_head"cache 用户加速访

问，buffer_init()）

19、创建页 cache（内存页 hash 表初始化，page_cache_init()）

20、创建信号队列 cache（"signal_queue"，signals_init()）

21、初始化内存 inode 表（inode_init()）

22、创建内存文件描述符表（"filp_cache"，file_table_init()）

23、启动 init 过程（创建第一个内核线程，调用 init()函数，原执行序列调用 cpu_idle() 等待调度，init()）

init()函数作为内核线程，首先锁定内核（仅对 SMP 机器有效），然后调用 do_basic_setup()完成外设及其驱动程序的加载和初始化。过程如下：

1、总线初始化（比如 pci_init()）

2、网络初始化（初始化网络数据结构，包括 sk_init()、skb_init()和 proto_init()三部分，在 proto_init()中，将调用 protocols 结构中包含的所有协议的初始化过程，sock_init()）

3、创建 bdflush 内核线程（bdflush()过程常驻内核空间，由内核唤醒来清理被写过的内存缓冲区，当 bdflush()由 kernel_thread()启动后，它将自己命名为 kflushd）

4、创建 kupdate 内核线程（kupdate()过程常驻内核空间，由内核按时调度执行，将内存缓冲区中的信息更新到磁盘中，更新的内容包括超级块和 inode 表）

5、设置并启动内核调页线程 kswapd(为了防止 kswapd 启动时将版本信息输出到其他信息中间，内核线调用 kswapd_setup()设置 kswapd 运行所要求的环境，然后再创建 kswapd 内核线程)

6、创建事件管理核心线程（start_context_thread()函数启动 context_thread()过程，并重命名为 keventd）

7、设备初始化（包括并口 parport_init()、字符设备 chr_dev_init()、块设备 blk_dev_init()、SCSI 设备 scsi_dev_init()、网络设备 net_dev_init()、磁盘初始化及分区检查等等，device_setup()）

8、执行文件格式设置（binfmt_setup()）

9、启动任何使用__initcall 标识的函数（方便内核开发者添加启动函数，do_initcalls()）

10、文件系统初始化（filesystem_setup()）

11、安装 root 文件系统（mount_root()）

至此 do_basic_setup()函数返回 init()，在释放启动内存段（free_initmem()）并给内核解锁以后，init()打开/dev/console 设备，重定向 stdin、stdout 和 stderr 到控制台，最后，搜索文件系统中的 init 程序（或者由 init=命令行参数指定的程序），并使用 execve()系统调用加载执行 init 程序。

init()函数到此结束，内核的引导部分也到此结束了，这个由 start_kernel()创建的第一个线程已经成为一个用户模式下的进程了。此时系统中存在着六个运行实体：

1、 start_kernel()本身所在的程序其实是一个手工创建的线程，它在创建了 init()线程以后就进入 cpu_idle()循环了，它不会在进程（线程）列表中出现

2、 init 线程，由 start_kernel()创建，当前处于用户态，加载了 init

程序

3、 kflushd 内核线程，由 init 线程创建，在内核态运行 bdflush()函数

4、 kupdate 内核线程，由 init 线程创建，在内核态运行 kupdate()函数

5、 kswapd 内核线程，由 init 线程创建，在内核态运行 kswapd()函数

6、 keventd 内核线程，由 init 线程创建，在内核态运行 context_thread()函数

init 进程是系统所有进程的起点，内核在完成核内引导以后，即在本线程（进程）空间内加载 init 程序，它的进程号是 1。

init 程序需要读取/etc/inittab 文件作为其行为指针，inittab 是以行为单位的描述性（非执行性）文本，每一个指令行都具有以下格式：

id:runlevel:action:process 其中 id 为入口标识符，runlevel 为运行级别，action 为动作代号，process 为具体的执行程序。

id 一般要求 4 个字符以内，对于 getty 或其他 login 程序项，要求 id 与 tty 的编号相同，否则 getty 程序将不能正常工作。

runlevel 是 init 所处于的运行级别的标识，一般使用 0－6 以及 S 或 s。0、1、6 运行级别被系统保留，0 作为 shutdown 动作，1 作为重启至单用户模式，6 为重启；S 和 s 意义相同，表示单用户模式，且无需 inittab 文件，因此也不在 inittab 中出现，实际上，进入单用户模式时，init 直接在控制台（/dev/console）上运行/sbin/sulogin。

在一般的系统实现中，都使用了 2、3、4、5 几个级别，在 Redhat 系统中，2 表示无 NFS 支持的多用户模式，3 表示完全多用户模式（也是最常用的级别），4 保留给用户自定义，5 表示 XDM 图形登录方式。7－9 级别也是可以使用的，传统的 Unix 系统没有定义这几个级别。runlevel 可以是并列的多个值，以匹配多个运行级别，对大多数 action 来说，仅当 runlevel 与当前运行级别匹配成功才会执行。

initdefault 是一个特殊的 action 值，用于标识缺省的启动级别；当 init 由内核激活以后，它将读取 inittab 中的 initdefault 项，取得其中的 runlevel，并作为当前的运行级别。如果没有 inittab 文件，或者其中没有 initdefault 项，init 将在控制台上请求输入 runlevel。

sysinit、boot、bootwait 等 action 将在系统启动时无条件运行，而忽略其中的 runlevel，其余的 action（不含 initdefault）都与某个 runlevel 相关。

在本文的研究过程所使用的系统中，使用的是 3 这个运行级别。 根据 StrongARM 上 Linux 的整个启动过程可以了解在 StrongARM 平台上启动系统，以及系统会需要那些具体的配置和硬件支持，有助于减少系统启动时间，提高系统的性能，并在系统启动错误时准确定位到相应的启动部分。

根据以上分析，针对 StrongARM 平台，本文对 Linux 的启动过程进行了如下改动：

1、在内核启动前，由 BootLoader 在寄存器中设置处理器类型，通知 Linux 内核启动平台类型。

2、本系统中采用的是混合式文件系统，使用 Flash 存储设备，在启动前设置内核参数，指定 Ramdisk 和 JFFS 文件系统的起始 Flash 块号(实现中 JFFS 使用的是”mtdblock6”块号)。

3、在 Linux 内核进行设备检测过程中，某些设备不会为嵌入式所使用，如 SCSI 设备、IDE 设备等，在设备检测阶段去除了相应的检测代码。

4、Linux 内核检测到网卡后，需要进行网络设备配置，在 Init 程序执行过程前，配置 rc.local 文件，进行网络设置。

针对 StrongARM 平台进行上述的启动过程改进后，在完成整个系统配置的基础上，系统启动时间得以大大缩短，系统效率也得到很大提高。

4.7.3 启动界面改进

Linux 在启动时会在屏幕的左上方显示一个启动画面，并在其下方显示启动的进程和设备的启动结果，对于嵌入式的很多应用而言，并不希望用户在系统启动时看到这些繁琐的启动信息，所以需要屏蔽这些对用户没有用处的启动信息，最好的办法就是当系统启动时，象很多手机、PDA 一样，给用户显示一个启动图片，掩盖相应的启动信息。

Linux 中可以通过 Frambuffer 机制屏蔽这些启动信息，在 Linux 内核中，如果支持 Frambuffer，启动时设置内核自动加载图片，但一般内核的图片很小，不足以掩盖所有的启动信息。所以需要重新加载图片。修改内核的一部分源码，Linux 下在 driver/video/fbcon.c 中 CONFIG_FBCON_LOGO_BOOT 定义了图片的大小，修改这个常量为屏幕的行数减一，让用户可以看到一行的启动信息，这行启动信息会随着系统启动不停的刷新。当然如果将 CONFIG_FBCON_LOGO_BOOT 定义为屏幕的行数，所有的启动系统都会被屏蔽掉。

然后制作新的启动图片，并使用 bmptoppm 工具将其转换为 ppm 格式的图片，然后使用 gcc 将其转换为 C 程序的头文件形式。最后在内核中选上 Cutom boot splash screen，重新编译内核，启动信息就可以被屏蔽掉了。实际系统的图片参看图 4-4

4.8 StrongARM 平台上的 LEOS 性能评估

在完成 LEOS 的实现后，本文针对一些重点问题进行了一系列测试，以获得系统的性能参数，对系统进行客观的评价。

4.8.1 实时内核调度时间精度测试

实时系统的重要指标之一就是对任务执行时间的可预测性，在硬实时系统中，由于调度误差主要来自定时硬件的限制，而在非实时系统中，无法估计的来自低优先级任务的阻塞可能使得调度误差恶化。

实验如下：在使用未实时化内核的和已实时化内核的两个系统下分别构造一个优先级最高的周期任务，在每个周期开始读取 TSC 寄存器获得精确的系统时间并纪录。通过改变任务周期和增加不同种类的低优先级任务,得到调度误差与环境的关系，如下表所示：

表 4-3 实时内核调度时间精度测试

	未实时化系统误差	已实时化系统误差
无其他负载的调度时间精度	14.4	0.623
轻负载的调度时间精度	174.2	1.683
重负载的调度时间精度	429.7	2.010

（单位：微秒）

上表显示了在三种环境下，非实时内核和实时内核中以 10ms 为周期调度实时任务的调度误差随负载任务的变化。可以看出系统负载较高时，普通 Linux 内核的不可抢占性使得实时任务调度时间的误差变化急剧增大。而实时化内核中大量非实时进程对调度精度的影响在微秒量级，已经接近硬件的极限。从而证明了系统在实时方面可以满足大多数的实时应用。

4.8.2 文件系统性能测试

LEOS 采用混合文件系统后，系统在各方面的性能都有很大提高，具体性能的测试如下：

测试一：分别使用混合式文件系统和 JFFS2 两种文件系统做为系统的根文件系统，获得两次的启动时间分别为：

使用混合式文件系统的系统启动时间为 9.941 秒。

使用 JFFS2 的系统启动时间为 25.020 秒。

测试二：重复创建并删除文件操作若干次，计算实际运行时间，测试原程序如下：

```
int function()
{
        FILE *fp;

        creat(FILENAME,S_IRWXU);
        unlink(FILENAME);
        return 0;
}
int main()
{
        struct timeval tpstart,tpend;
        float  timeuse = 0;
        int i;


        for(i=0; i<TESTTIMES; i++){

                gettimeofday(&tpstart,NULL);
                function();
                gettimeofday(&tpend,NULL);
```

```
timeuse+=1000000*(tpend.tv_sec-tpstart.tv_sec)+
tpend.tv_usec-tpstart.tv_usec;
        }

        timeuse/=1000000;
        printf("To Exec %d Times Function Used Time:%f
Average: %f \n",TESTTIMES,timeuse,timeuse/TESTTIMES);

        return 0;
}
```

其中 TESTTIMES 即为测试项 function 的测试次数，结果如下：

表 4-4 文件系统性能测试结果

测试次数	**PC**	混合式文件系统	**JFFS2**
100	0.002668/0.000027	0.020479 / 0.000205	0.891897/0.008919
1000	0.027866/0.000028	0.221584 / 0.000222	7.018943/0.007019

（总时间/平均时间，单位：秒）

测试结果分析：

从测试一可以看出，JFFS2 在启动时由于需要扫描整个 FLASH 介质，所以需要花费很长时间用于系统启动，而混合式文件系统系统直接从 Flash 中读取 RAMDISK 文件系统到内存中，不需要进行 Flash 的扫描，所以启动时间大大缩短，大约为 JFFS2 系统的 1/3 强。

测试二中，从结果来看混合式文件系统的文件创建和删除时间大约是 JFFS2 的 1/40，性能有极大的提高。

4.8.3 中文显示功能测试

由于 Linux 内核在设计时没有考虑到国际化的问题，使得做为中文用户，在使用时无法获得内核的中文支持。本文通过分析 Linux 的显示机制，完成了 Linux 内核对于中文的支持，并为内核加入了中文字库，使得 Linux 在内核级支持中文显示。由于 Linux 在启动后会直接显示其内核版本信息，所以测试方案如下：

在 Linux 内核源码的版本信息文件"/include/linux/version.h"中，修改其版本信息为：

```
#define UTS_RELEASE "2.4.17-rmk5-中科院软件所"
```

内核中文化后应可以正常显示版本里面的中文字符。测试结果如图 4-4 所示，中文显示正常。

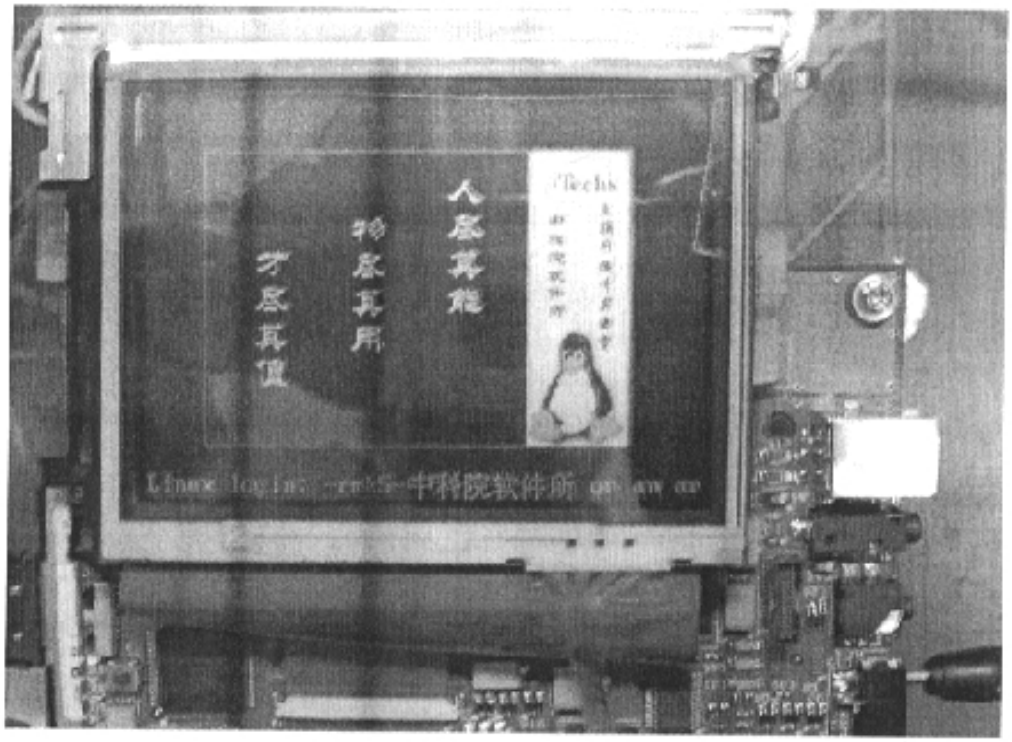

图 4-4 LEOS 启动界面图

4.8.4 系统评价

本章设计并实现了 StrongARM 平台上的 LEOS，除了提供标准 Linux 的基本功能，如进程管理、内存管理、中断管理、设备管理和底层网络服务等功能之外，还在此基础上增加了面向嵌入式的关键技术改进，包括：内核实时化、混合式文件系统和内核中文处理、。

在 StrongARM 平台上的实现还包括其上的各种设备支持，在实现过程中针对嵌入式资源和空间有限的特点，进行了大量的改进和提高。从系统功能的完整性和可靠性上，已与通常的商用嵌入式操作系统如 Windows CE 等相似，能够基本满足应用层的各种需求，但是在人机界面和开发环境等方面与嵌入式商用系统还有一定差距。

从以上的分析和前面的测试内容可以看出，LEOS 能够较好的满足大多数的嵌入式应用需求，提供较为完整的操作系统底层支持，在此基础上能够为嵌入式应用和开发提供基本的服务和支持。并且由于 Linux 是开放源码操作系统，使得本文的研究成果具有较高的应用价值和一定的学术意义。

4.9 小结

本章主要介绍了在 StrongARM 平台上建立 LEOS 及其相关的工作，包括实现实时内核的构建和双内核体系，建立和优化文件系统，内核的中文化支持，以及针对 StrongARM 平台的内核改进。并提出了完整的 LEOS 的解决方案，目的是为嵌入式应用构建一个通用的 LEOS 平台，在其上运行通常的嵌入式应用。LEOS 通过改善、优化和扩展可应用于其它有特定要求的领域。

第五章 结束语

5.1 本文工作总结

嵌入式技术有着广泛的应用前景，其技术发展非常迅速。操作系统是嵌入式系统的基础，它的功能和效率决定了嵌入式应用的性能，因此嵌入式操作系统是嵌入式系统的核心技术之一。

本文从嵌入式系统的特性着手，分析了目前嵌入式操作系统的现状和主流的产品，在此基础上设计并在 StrongARM 平台上实现了 LEOS，主要研究成果包括：

1、根据国内当前嵌入式应用的特定需要，设计了 LEOS 的整体解决方案。
2、重点研究了双内核的实时化方案、混合式文件系统方案和内核中文化方案。解决了 Linux 应用于嵌入式系统的关键技术问题。
3、基于以上设计，在 Intel 公司的 StrongARM 嵌入式硬件平台上开发实现了 LEOS，实现了设计中的所提出的方案，并针对该平台进行了相应的优化。

LEOS 能够具有一定的实时性能，采用了高效的混合式文件系统，支持中文显示，这些特性使得其能够满足大多数的嵌入式应用的基本需求。目前已经开始在这个平台上开发一些具体的嵌入式应用，主要是网络和终端应用，该嵌入式系统能够为这些应用提供很好的支持，可见通过本文的研究，为嵌入式应用提供了良好的支持环境。可以预见，在不远的将来，一定可以带来良好的应用前景。

5.2 进一步的工作

为了能够更好的为不同嵌入式应用提供系统级别的支持，现有的 LEOS 还需要进一步完善。

进一步的工作可以在现有系统的基础上，对系统的电源管理、多种设备支持等方面进行改进，提高系统的性能，满足更多的应用要求。

参考文献

[1] Institution of Electrical Engineers, *URL:http://www.IEE.org*

[2] D. Desmet, D. Verkest, and H. De Man, "Operating System Based Software Generation for Systems-on-Chip", *Proc. Design Automation Conf., June 2000.*

[3] B. Lin, S. Vercauteren, H. De Man, "Embedded architecture co-synthesis and system integration", *Fourth International Workshop on Hardware/Software Co-Design, Pittsburgh, PA, pp. 2-9, 1996.*

[4] D. D. Gajski and F. Vahid, "Specification and design of embedded software-hardware systems," *IEEE Design & Test of Computers, vol. 12, no. 1, Spring 1995.*

[5] 王飞跃，吴朝晖，ASOS：嵌入式操作系统的发展趋势，URL：*http://www.ccw.com.cn/htm/produ/corner/0111_2.asp, 2003.6*

[6] 钟锡昌，嵌入操作系统在中国的发展现状与前景，URL：*http://www.embedded-tech.com.cn/meet2002/jh6.asp, 2002.3*

[7] Linux, *URL:http://www.linux.org*

[8] Darrick Addison，“嵌入式 Linux 应用：概述”，August 2001，URL：*http://www-900.ibm.com/developerWorks/cn/linux/ embed/embl/overview/index.shtml* .

[9] MontaVista, *URL:http://www.mvista.com/*

[10] ucLinux, *URL:http://www.uclinux.com*

[11] Lynuxworks, *URL:http://www.lynuxworks.com*

[12] Intel® StrongARM* SA-1110 Microprocessor Brief Datasheet, *Intel Corporation.*

[13] The Arm Linux Project. *URL:http://www.arm.linux.org.uk/*

[14] R. Witek, J.Montanaro, StrongARM: a high-performance ARM processor, *COMPCON Spring '96 - 41st IEEE International Computer Conference, San Jose, CA,* 1996

[15] 贾小涛，陈章龙，“嵌入式处理器 StrongARM 的开发研究”，*计算机工程*，2002(08)。

[16] 李云飞，胡剑凌，陈健，“基于 StrongARM 的远程网络监控系统设计”，*计算机应用*，2002(11)。

[17] Intel® StrongARM* SA-1110 Microprocessor Development Board User's Guide, *Intel Corporation.*

[18] Intel® StrongARM* SA-1110 Microprocessor Developer's Manual, *Intel Corporation.*

[19] Cort Dougran, Linux on the PowerPC: Optimizing Modern Operating System for Modern Processor, *New Mexico Institute of Mining and Technology,* 1998.

[20] Michael S., Focusing Real-Time Systems Analysis on User

Operations, *IEEE Software*, 1988 5(5)

[21] KARSTEN SCHWAN, TOM BIHARI, BRUCE W. WEIDE, GREGOR TAULBEE, High-Performance Operating System Primitives for Robotics and Real Time Control Systems, *ACM Transactions on Computer Systems, Vol. 5, No. 3, August 1987.*

[22] 蔡君，Linux 的冲击和操作系统的革命，2000.5，*互联网世界*

[23] 钟汉如，王创生，“嵌入式 Linux 的中断处理与实时调度的实现机制”，*计算机工程*，2002(10)

[24] 邢国良，韦宏利，伍卫国，陈剑，“基于 Linux 的实时操作系统的分析与研究”，*小型微型计算机，2001,22(8)*

[25] E.Bianchi, L.Dozio, P.Mantegazza, Real Time Application Interface (for Linux) A Hard Real Time support for LINUX, *Dipartimento di Ingegneria Aerospaziale, Politecnico di Milano*

[26] DIAPM RTAI Programming Guide 1.0, *Lineo, Inc.*

[27] Marty Humphrey, Edgar Hilton, Paul Allaire, Experiences Using RT-Linux to Implement a Controller for a High Speed Magnetic Bearing System, *Fifth IEEE Real-Time Technology and Applications Symposium, Canada,* 1999

[28] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System," Proceedings of the 1995 USENIX Technical Conference, Jan. 1995.

[29] Understanding the Flash Translation Layer (FTL) Specification, *Intel Corporation.*

[30] David Woodhouse, JFFS : The Journalling Flash File System, *Red Hat, Inc.*

[31] Daniel P. Bovet, Marco Cesati, Understanding the Linux Kernel(Second Edition), *O'Reilly,* 2002.

[32] Linux Development on the SA1110/1111 or SA1100 Platform, *Intel Corporation.*

[33] Intel® StrongARM* SA-1110/SA-1111 Development Kit Quick Start Procedures User's Guide, *Intel Corporation.*

[34] RedBoot™ User's Guide, *Red Hat, Inc.*

[35] 李善平，刘文峰，王焕龙，“Linux 与嵌入式系统”，*清华大学出版社*，2003。

发表文章目录

王巍, 王青, 李明树 《非冲突的实时以太网协议研究与实现》 中科院研究生学术研讨会, 2002-07

致谢

感谢我的导师王青研究员和李明树研究员，在我的三年研究生生活中，两位老师为我创造了宽松的学术环境，并在学术研究方向和方法上给予了悉心的指导，同时他们也在为人处事方面严格要求，两位老师的严谨的作风，让我不仅在学术上有所进步，更在做人方面受益匪浅。

感谢周津慧老师、王永吉老师和李怀璋老师，他们在我碰到困难时，给予了我极大的帮助，不仅有学术上也有生活上的帮助，让我感到了实验室集体的温暖。

感谢实验室实时操作系统组的全体成员：史兴国、杨立光、孙岩、余圳、舒国强、梁松、杨小虎、李润博、王绍恒、吴立国、杨达、林溯弈等领导、同事和同学，我的论文是在大家的工作积累基础上完成的，我深深感到，我们是一个气氛融洽、富有合作精神的团队，非常怀念一起工作的日子。

感谢实验室其他成员：张晓刚、李雄锋、张津诗等。

感谢我的父亲、母亲和女友对我的支持、关心和鼓励，他们是我永远乐观向上的精神支柱。

再次感谢所有给予我帮助的良师益友。