摘 要

Linux操作系统优异的可靠性、良好的可裁减性、广泛的技术支持，受到了技术界的推崇和赞许，并在许多产品中得到了大量地使用。现在它已经支持几乎所有主流的32位CPU，新的2.6版内核更提供了许多针对嵌入式应用的支持，并且改进了进程调试算法。使得Linux在嵌入式系统中的应用备受关注，目前正被手持设备如手机、导航仪等信息终端领域获得广泛应用。而嵌入式操作系统的关键技术之一，便是板级支持软件(BSP,Board Suport Package)的实现。

本文采用 SAMSUNG 公司的 32 位 S3C2440 处理器作为导航系统的硬件平台的核心处理单元，研究并设计了支持 nandflash 启动，支持多文件系统的 BSP，提出了若干种BSP的优化方法，并在此基础上利用GPS导航、GPRS、嵌入式S3C2440处理器和嵌入式 Linux 操作系统的优良特性，设计了适合嵌入式导航系统的 BSP 软件系统。本文主要从 BSP 的组成结构，执行流程，设计思路，模型设计等方面进行了研究和探讨。着重分析和研究了 BSP 的引导代码设计，操作系统内核移植以及驱动程序框架和设计，并在此基础上针对导航系统特定应用优化了 BSP 软件系统。论文完成的主要工作有如下几点：

（1）研究了 boot loader 的工作原理，并根据嵌入式导航系统的实际硬件资源的特点，实现了引导代码 boot loader 的设计，使它在目标板上电后完成硬件初始化、下载和引导内核的工作，并为应用开发人员烧写操作系统内核或文件系统设计了一个良好的人机交互界面。为了验证设计的引导程序的正确性，本文将 boot loader 编译成二进制代码，烧入 Nandflash 启动运行，对其功能进行了详细的测试和验证。

（2）完成了 Linux 操作系统向嵌入式导航平台的移植，包括内核的定制与片上文件系统的设计。针对嵌入式导航系统要求文件系统占用存储空间少，系统性能好，且可写的特点，本文提出了使用 Squanshfs 文件系统，可写的 yaffs2 文件系统和 tmpfs 文件系统组合的方法，充分利用多个文件系统的优点，满足了系统的需求。并将内核和文件系统生成映像文件，通过 boot loader 的下载功能烧入 Nandflash 启动运行，对内核与文件系统的运行情况进行了测试。

（3）通过大量驱动案例分析，研究了各类驱动的层次和接口，给出了设备驱动程序设计框架，并完成了实现导航系统的USB驱动、触摸屏驱动和串口驱动的驱动程序的设计与开发。

（4）提出了若干种改善系统启动速度和占用空间的优化方法，并给出了具体的实现技术和过程，满足了嵌入式系统启动速度快，存储空间少的要求。

最后，将整个优化设计出的BSP软件应用于嵌入式智能交通导航系统中，作为嵌入式导航系统的基础软件，在本BSP的基础上，配合上层应用软件，已实现了实际嵌入式智能交通导航设备。通过智能交通导航系统来验证了本BSP的优异性能。本BSP也可应用于其他类似的嵌入式应用系统中。

关键字： GPS导航、Linux移植、系统启动优化、引导程序、设备驱动

ABSTRACT

Linux is very excellent operation system. We can get source files of it from INTERNET without charge. Because it is very easy to configure, reduce and well Supported, Linux is used in very wide fields now. Especial in the fields of embedded system and it became the most popular embedded operation system. Now embedded Linux was used in many kinds of portable terminal produces such as mobile, navigation device and so on. BSP (Board Support Package) is one of the key technologies of embedded Linux.

This paper used SAMSUNG's S3C2440 processor 32 as navigation system hardware platform core process unit, researched and designed a BSP which support nandflash startup and multi-file system, brought forward a number of optimization methods about BSP, and using GPS navigation, GPRS, embedded S3C2440 processor and embedded Linux operating system fine characteristics, designed the BSP software systems which was suitable for embedded navigation system. In this paper, studied and discussed BSP from composition of the structure, implementation of process, design ideas, models design. Analyzed and studied BSP's boot-up code design, operating system kernel porting, and driver framework and design ideas, on this basis, for navigation system specific application optimized software system BSP. Paper will complete primary tasks as follows:

(1) This paper studied the principle of the boot loader, according to actual characteristics of embedded navigation system's hardware resources, realized the design of boot loader boot-up code. When the target board power on, completed hardware initialization, downloads and boot-up kernel functions, Designed a good interactive interface to program kernel or file system into nandflash for application developers. In order to verify the correctness of boot, boot loader will be compiled into binary code, and program into Nandflash, start running, its function will be test in detail.

(2) This paper completed Linux operating system porting to embedded navigation platform, including customized kernel and file system design. Navigation system required embedded file systems occupy less storage space, have a good performance, and can be written, this paper presents to combinate Squanshfs, yaffs2 and tmpfs file system into a whole, make full use of multiple file systems' advantages to meet the requirements. And make kernel and file system into image program into nandflash through download function of boot loader. Test kernel and file system performance.

(3) Through analyze a large number of case, studied a variety of drivers hierarchy and interfaces. Given the design of the device driver framework, completed the USB drive, touch screen driver, serial driver of navigation system.

(4) Brought forward a number of optimization methods to improve the speed of startup and reduce storage space, given the method and process. meet the embedded system requirement that boot faster, less storage space.

Finally, the whole BSP software optimized design was applied in embedded intelligent traffic navigation system, take use of foundation software. On the basis of BSP, cooperate with application, realized embedded intelligent traffic navigation system devices. Through the intelligent traffic Navigation System to verify the excellent performance of the BSP. The BSP can also be applied to other similar systems in embedded applications.

Key Words: GPS navigation, Linux porting， System Bootup Optimization, Bootup Program, Device driver

独创性声明

秉承学校严谨的学风与优良的科学道德，本人声明所呈交的论文是我个人在导师的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的地方外，论文中不包含其他人已经发表或撰写过的研究成果，不包含本人或其他用途使用过的成果。与我一同工作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，并表示了谢意。

本学位论文成果是本人在广东工业大学读书期间在导师的指导下取得的，论文成果归广东工业大学所有。

申请学位论文与资料若有不实之处，本人承担一切相关责任，特此声明。

论文作者签字：

指导教师签字：

年　月　日

第一章 绪论

1.1 课题研究背景

导航系统采用了全球卫星定位系统(Global Position System)，简称 GPS 导航技术。GPS 卫星导航定位技术是一项高新技术，它利用了 GPS 接收导航卫星的发射信号，从而获取当前位置的坐标和时间等信息，达到定位、导航和测量的目的。卫星导航定位技术被广泛应用在社会的各个领域中，它的高精度、快捷方便、全天候等优良特性，使其越来越受到人们的欢迎[01]。

GPS 系统是一个高精度的定位导航系统。利用 GPS 导航定位的各种移动终端(如手机、掌上电脑等)实质上就是一个嵌入式系统。在这种作为移动终端的嵌入式系统上开发 GPS 导航软件的方式主要有两种：一种是在自己开发的专用的操作系统上开发导航软件；另一种是在通用的操作系统上开发导航软件。日本及欧美等地的发达国家起步较早，二十世纪八十年代就开始开发导航软件，采用的是前一种开发方式，有自己的硬件设备，底层的操作系统也由自己开发，并基于其上开发导航软件，这种开发方式由于其专用性，数据的保密性较好，但通用性不好，仅可用于有限的嵌入式设备上[02]。而本文则是采用后一种方式开发。在通用的 Linux 操作系统上开发导航系统。

1.1.1 嵌入式系统现状及特点

嵌入式系统就是嵌入式计算机系统的简称。IEEE(国际电气和电子工程师协会)对它的定义是“Device used to control, monitor or assist the operation of equipment, machinery or plants"(用于控制、监视功能或辅助仪器、机械、设备上作的装置)。一般常用的定义是：嵌入式系统指非 PC 系统，有计算机功能但又不称之为计算机的设备或器材，即不可见的计算机。它是以应用为中心，软硬件可裁减的适应应用系统对功能、可靠性、成本、体积、功耗等综合性严格要求的专用计算机系统。简单地说，嵌入式系统集应用软件与硬件于一体，具有软件代码小、高度自动化、响应速度快等特点，特别适合于要求实时和多任务的体系[06]。

如图 1-1 所示，嵌入式系统的组成一般由硬件和软件组成。嵌入式硬件以嵌入式处理器为核心集成存储器和系统专用的输入/输出设备。嵌入式软件包括固件、嵌入式操作系统和应用程序等，这些软件有机的结合在一起，形成系统特定的一体化软件[07]。

上世纪 80 年代随着集成电路技术的飞速发展，嵌入式应用领域不断扩大，应用要求不断提高，在实时性、可靠性、多任务等方面提出了越来越多的要求，系统已经不能用简单的循环控制处理了，在嵌入式系统中引入操作系统成为必然。因此，在现代嵌入式系统一般都包括了嵌入式操作系统，它是嵌入式应用软件与嵌入式硬件之间的管理者和协调者。

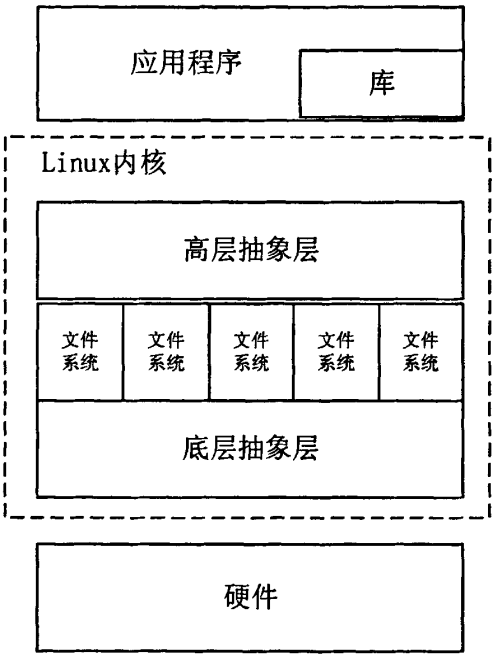


图 1-1 嵌入式系统的一般框架

Figure 1-1 General framework for embedded systems

在符合系统要求的情况下，嵌入式系统应尽量减少不必要的外设，减小系统功耗。一般嵌入式系统由以下几个模块组成：

- 微控制器，通常是 4 位、8 位、16 位、32 位或 64 位。
- 用以保存固件的 ROM (非挥发性只读存储器)。
- 用以保存程序数据的 RAM (挥发性的随机访问存储器)。
- 周围相关硬件，如 LED(发光二极管)、串口、网口、传感器、开关等。
- 嵌入式操作系统。

嵌入式系统的一般模型并不足以定义嵌入式系统本身。例如:

- 信息查询以及销售点终端。
- 某些工业控制系统。
- 游戏控制台 (例如基于 X86 和 Windows 的 Xbox) 。
- 数字录像机 (例如基于 Linux 的 TiVo) 。

这些设备可能使用硬盘驱动器来保存固件，并运行各种桌面操作系统如 Windows， Linux 或者 DOS。这并不是典型的嵌入式系统的体系结构，但是按照定义，它们仍可以称为嵌入式系统。

嵌入式系统具有专用性、高可靠性、可封装性等特点。它的发展历史几乎和计算机自身的历史一样长，发展过程大致分成以下 4 个阶段[08]。

第一阶段是以单芯片为核心的可编程控制器形式的系统，同时具有与监测、伺服、指示设备相配合的功能。这种系统大部分应用于一些专业性极强的工业控制系统中，一般没有操作系统的支持，通过汇编语言编程对系统进行直接控制，运行结束后清除内存。这一阶段系统的主要特点是：系统结构和功能都相对单一，处理效率较低，存储容量较小，几乎没有用户接口。由于这种嵌入式系统使用简便、价格很低，以前在国内工业领域应用较为普遍，但是已经远远不能适应高效的、需要大容量存储介质的现代化工业控制和新兴的信息家电等领域的需求。

第二阶段是以嵌入式 CPU 为基础、以简单操作系统为核心的嵌入式系统。这一阶段系统的主要特点是：CPU 种类繁多，通用性比较弱；系统开销小，效率高；操作系统具有一定的兼容性和扩展性：应用软件较专业，用户界面不够友好，系统主要用来控制系统负载以及监控应用程序运行。

第三阶段是以嵌入式操作系统为标志的嵌入式系统。这一阶段系统的主要特点是：嵌入式操作系统能运行于各种不同类型的微处理器上，兼容性好；操作系统内核精小、效率高，并且具有高度的模块化和扩展性；具备文件和目录管理、设备支持、多任务、网络支持、图形窗口以及用户界面等功能；具有大量的应用程序接口(API)，开发应用程序简单；嵌入式应用软件丰富。

第四阶段是以基于 Intenret 为标志的嵌入式系统，这是一个正在迅速发展的阶段。目前大多数嵌入式系统还孤立于 Internet 之外，但随着 Internet 的发展以及 Intemel 技术与信息家电，工业控制技术等结合日益密切，嵌入式设备与 Internet 的结合将代表着嵌入式技术的真正未来。

1.1.1.1 嵌入式处理器的发展现状

嵌入式系统的核心部件是各种类型的嵌入式处理器，目前据不完全统计，全世界嵌入式处理器的品种总量已经超过1000多种，流行体系结构有30几个系列，其中8051体系的占有多半。生产8051单片机的半导体厂家有20多个，共350多种衍生产品，仅Philips就有近100种。现在几乎每个半导体制造商都生产嵌入式处理器，越来越多的公司有自己的处理器设计部门。嵌入式处理器的寻址空间一般从64kB到16MB，处理速度从0.1 MIPS到2000 MIPS，常用封装从8个引脚到144个引脚。根据其现状，嵌入式计算机可以分下面几类：

- 嵌入式微处理器（Embedded Micorporcessor Unit，EMPU）

嵌入式微处理器的基础是通用计算机中的CPU。在应用中，将微处理器装配在专门设计的电路板上，只保留与嵌入式应用有关的母板功能，这样可以大幅度减小系统体积和功耗。为了满足嵌入式应用的特殊要求，嵌入式微处理器虽然在功能上和标准微处理器基本是一样的，但在工作温度、抗电磁干扰、可靠性等方面一般都做了各种增强。

和工业控制计算机相比，嵌入式微处理器具有体积小、重量轻、成本低、可靠性高的优点，但是在电路板上必须包括ROM、RAM、总线接口、各种外设等器件，从而降低了系统的可靠性，技术保密性也较差。嵌入式微处理器及其存储器、总线、外设等安装在一块电路板上，称为单板计算机。如STD-BUS、PC104等。近年来，德国、日本的一些公司又开发出了类似“火柴盒”式名片大小的嵌入式计算机系列OEM产品[09]。

嵌入式处理器目前主要有Am186/88、386EX、SC-400、Power PC、68000、MIPS、 ARM系列等。

- 嵌入式微控制器(Microcontroller Unit，MCU)

嵌入式微控制器又称单片机。顾名思义，就是将整个计算机系统集成到一块芯片中。嵌入式微控制器一般以某一种微处理器内核为核心，芯片内部集成ROM/EPROM、RAM、总线、总线逻辑、定时/计数器、WatchDog、FO、串行口、脉宽调制输出、A/D、D/A、 Flash RAM、EEPROM等各种必要功能和外设。为适应不同的应用需求，一般一个系列的单片机具有多种衍生产品，每种衍生产品的处理器内核都是一样的，不同的是存储器和外设的配置及封装。这样可以使单片机最大限度地和应用需求相匹配，功能不多不少，从而减少功耗和成本。

和嵌入式微处理器相比，微控制器的最大特点是单片化，体积大大减小，从而使功耗和成本下降，可靠性提高。微控制器是目前嵌入式系统工业的主流。微控制器的片上外设资源一般比较丰富，适合于控制，因此称为微控制器[10]。

嵌入式微控制器目前的品种和数量最多，比较有代表性的通用系列包括8051、P51XA、MCS-251、MCS-96/196/296、C166/167、MC68HC05/11/12/16、68300等。另外还有许多半通用系列如：支持USB接口的MCU 8XC930/931、C540、C541，支持I2C、CAN-Bus、LCD及众多专用MCU和兼容系列。目前MCU占嵌入式系统约70%的市场份额。

特别值得注意的是近年来提供X86微处理器的著名厂商AMD公司，将Am186CC/CIFCU等嵌入式处理器称之为Microcontroller，MOTOROLA公司把以Power PC为基础的PPC505和PPC555亦列入单片机行列。TI公司亦将其TMS320C2XXX系列DSP作为MCU进行推广。

- 嵌入式片上系统（System On Chip）

随着EDI的推广和VLSI设计的普及化及半导体工艺的迅速发展，在一个硅片上实现一个更为复杂的系统的时代已来临，这就是System On Chip（SOC）。各种通用处理器内核将作为SOC设计公司的标准库。和许多其它嵌入式系统外设一样，成为VLSI设计中一种标准的器件，用标准的VHDL等语言描述，存储在器件库中。用户只需定义出其整个应用系统，仿真通过后就可以将设计图交给半导体工厂制作样品。这样除个别无法集成的器件以外，整个嵌入式系统大部分均可集成到一块或几块芯片中去，应用系统电路板将变得很简洁，对于减小体积和功耗，提高可靠性非常有利。

SOC可以分为通用和专用两类。通用系列包括Infincon的TriCore、Motorola的M-Core、某些ARM系列器件、Echelon和Motorola联合研制的Neuron芯片等。专用SOC一般专用于某个或某类系统中，不为一般用户所知。一个有代表性的产品是Philips的Smart XA，它将XA单片机内核和支持超过2048位复杂RSA算法的CCU单元制作在一块硅片上，形成一个可加载JAVA或C语言的专用的SOC，可用于公众互联网如Intemel安全方面。

1.1.1.2 嵌入式操作系统发展现状

随着微处理器技术和材料科学技术的迅猛发展，功能强大，价格低廉，结构小巧的 CPU 和外部设备提供了稳定可靠的硬件条件，限制嵌入式系统发展的瓶颈就突出表现在了软件方面，尤其需要强大的操作系统和相应的开发工具的支持。尽管从八十年代末开始，陆续出现了一些嵌入式操作系统，比较著名的有 Vxwork、pSOS、Neculeus 和 Windows CE。但这些专用操作系统都是商业化产品，其高昂的价格使许多公司望而却步。这些商用系统的源代码封闭性也大大限制了开发者的积极性，开发者得不到技术方面的强劲支持。同时，由于其代码的封闭性，对相关新的硬件设备的支持跟不上市场变化，因此影响基于该操作系统的产品开发，耽误了产品进入市场的最佳时机。现在需要的是一个便宜、成熟稳定并且能够提供高端嵌入式系统所必需特性的操作系统。嵌入式 Linux 操作系统以其稳定、高效、易定制、易裁减、硬件支持广泛、价格低等特点赢得广泛关注，成为新兴的力量，有成为嵌入式操作系统主导力量的趋势[08]。

特别是近几年来，随着开源软件开发队伍的壮大，以及一些跨国公司的加盟，Linux 在性能提高和应用广泛性等方面得到了迅速发展。2004 年 LinuxDevices.com 进行的市场调查显示，嵌入式 Linux 操作系统在过去 2 年里已经占到了嵌入式操作系统市场的 37%，而且预计未来两年中将达到 50%的占有率[11]。

1.1.2 嵌入式导航技术现状及特点

智能交通导航系统采用了全球卫星定位系统导航技术，GPS 卫星导航定位技术利用了 GPS 接收导航卫星的发射信号，从而获取当前位置的坐标和时间等信息，达到定位、导航和测量的目的。以 GPS 和电子地图为技术核心，为用户提供安全驾驶所必需的地图和道路等信息的交通导航系统目前正在全世界得到广泛应用，大大提高了交通导航技术与道路驾驶的安全性[03]。

近年来，为了对机动性强、数量众多的移动目标进行有效监视、紧急救援和提供各种信息服务的需求，在客运、公安、银行、物流等行业表现得尤为突出。通用分组无线业务(GPRS)的出现，使得人们能够对移动目标进行全国范围实时全天候的监视调度。将 GPS 定位技术和 GPRS 相结合，在智能交通导航方面具有广泛的应用前景[04]。

现在通用的流行的导航系统嵌入式平台一般有单片机系统平台、基于 ARM7 的嵌入式平台、基于 ARM9 S3C2410 的嵌入式平台。单片机电子地图应用系统信息处理能力弱、资源有限、图形界面实现较难、人机界面不友好，现在很少使用。基于 ARM7 的嵌入式平台资源比单片机系统要丰富一些，支持操作系统，在过去的定位系统和通信系统中用的很多，但对导航应用系统和复杂的图形界面，ARM7 处理速度慢。随着 ARM 技术的进步，ARM7 逐渐退出了导航领域。现在很多导航产品都采用基于 ARM9 S3C2410 的嵌入式平台，ARM9 通过提高时钟频率，改进指令周期，增加内存管理，在性能上比 ARM7 有了很大的提高。通过移植操作系统、图形库和触摸屏驱动，能实现友好的人机交互界面，能完成一般的导航任务。由于 S3C2410 主频是 266MHz，在运行导航系统时，系统响应不及时，在路径规划时等待时间过长[05]。所以，以上三种嵌入式平台都不能满足当前嵌入式导航系统的要求，迫切需要一种新的设计方案。

1.1.3 BSP 技术现状

板级支持包(Board Support Package，简称 BSP)技术一般来说是针对某个特定的嵌入式系统的，由嵌入式应用系统开发平台供应商提供，针对标准硬件板的各种驱动支持库。每个 BSP 包括一种软件模板，其中不仅包括设备驱动程序的抽象结构代码，而且包括具体硬件设备提供的底层硬件代码，还包括移植好的操作系统。板级支持包是嵌入式开发的关键环节[14]。有了稳定的 BSP，应用开发人员只要一心开发自己的应用程序就可以了，不需要关心与硬件相关的代码。这样给应用开发程序的开发带来了很大的便利。

1.1.3.1 BSP 的作用

目前，嵌入式系统开发大致可分为两个层次：

（1）嵌入式应用软件程序开发（主要利用 C 库函数和 Linux API 进行应用软件的编写）。

（2） 嵌入式 BSP 包的开发也就是固件的开发（主要进行 Boot loader、Linux 的移植及 Linux 设备驱动程序的设计）。

一般而言，BSP 包的开发的难度要高于应用程序开发的难度，而其中的 Linux 设备驱动编程又是 Linux 程序设计中比较复杂的部分。究其原因，主要包括如下几个方面：

（1）设备驱动属于 Linux 内核的部分，编写 Linux 设备驱动需要有一定的 Linux 操作系统内核基础。

（2）编写 Linux 设备驱动需要对硬件的原理有相当的了解，大多数情况下我们是针对一个特定的嵌入式硬件平台编写驱动的。

（3）Linux 设备驱动中广泛涉及到多进程并发的同步、互斥等控制，容易出现 bug。

（4）由于属于内核的一部分，Linux 设备驱动的调试也相当复杂。

所以 BSP 的开发，在整个嵌入式系统开发中起着关键的作用，整个 BSP 包的稳定性、运行速度、文件的大小对嵌入式产品是否成功起着很大作用[15]。

1.1.3.2 嵌入式应用系统 BSP 实现方式的比较

BSP 作为操作系统与硬件之间的一个桥梁，它既需要考虑对硬件的控制和处理，又需要涉及操作系统的调用接口和相关支持。因此 BSP 实现受到多方面的制约和影响。一般来说可以将 BSP 的技术路线描述为以下两种方式[17]：

（1）各功能函数和硬件驱动程序对上层应用不完全透明，应用程序对硬件的控制和操作将通过直接调用其驱动程序特定的操作函数来完成。在这种方式下，应用程序可以与硬件驱动程序相互交叉，并以一个完整的运行程序放到操作系统上运行。

（2）各功能函数和硬件驱动程序对上层应用透明、所有驱动程序由操作系统的管理、应用程序需要使用和控制的硬件等都必须通过操作系统的统一调用接口来完成调用。在这种方式下，应用程序与硬件驱动程序之间被隔离，需要通过操作系统的 API 接口才可能访问到相关的硬件，应用程序根本看不到驱动程序的存在，驱动程序与应用程序分别加载并在系统上运行。这种方式也就是通常所说的硬件抽象。

这两种方式实现与操作系统对驱动程序的管理和运行模式有一定的关系，这里暂不考虑操作系统差异的影响，主要讨论两种实现方式的主要优缺点。在这两

种方式下，BSP 主要区别体现在设备驱动程序与应用程序和操作系统之间的关系上，它们主要优缺点大致归纳如下：

第一种方式中，驱动程序在整个系统中注册和登记过程十分简单，实现也相对容易。而且由于应用程序可以直接调用驱动程序函数，因此可以减省管理环节和空间开销，以及应用程序与系统和驱动程序切换的开销，使得效率得到更好保证。但这种方式下，应用程序直接访问设备驱动程序，需要应用程序开发者对硬件有一定的了解和掌握，并能够准确控制硬件的操作。驱动程序的改变会引起应用程序的调整或重新编译。由于驱动程序不在操作系统控制下统一调度和使用，应用程序对外设的使用容易出现冲突和不稳定，从而有可能导致系统性能受到影响。

第二种方式中，操作系统对设备驱动程序进行管理和调度，并提供标准 API 接口。应用程序使用驱动程序控制硬件，都必须通过操作系统的 API 接口才能完成。这使得驱动程序对于应用程序来说是不可见的。硬件及其驱动程序的变化，不会影响应用程序设计及其对硬件的使用。这种透明性使得程序开发十分简便，也使得整个软件系统的移植操作变得十分高效。不同的硬件板，只要有相应的 BSP 加入到系统中，操作系统和应用软件就都可以在该硬件板上运行。当然以这种方式实现的 BSP 也存在一些缺点：通过系统调用访问设备驱动程序，由于应用程序往往工作于用户态，系统则工作于核心态。应用程序使用 API 接口时将引起用户态向核心态的切换，而这种操作的开销较大，会影响系统效率。另外在这种方式下设备驱动程序的开发需要按照操作系统的特定格式要求和注册流程进行设计，因此需要驱动程序开发者对操作系统的驱动程序体系结构有相当的认识[16]。

1.2 课题的研究内容与论文结构安排

1.2.1 课题的主要研究内容

课题研究并设计了一种支持 nandflash 启动，支持多文件系统的 BSP 软件，提出了几种 BSP 开发的优化方法，并在此基础上利用 GPS 导航、GPRS、嵌入式 S3C2440 处理器和嵌入式 Linux 操作系统的优良特性，设计了适合嵌入式导航系统的 BSP 软件系统。

在 BSP (Board Support Package)软件设计中，需要进行三个部分的工作——引导程序的设计与实现、嵌入式操作系统移植以及设备驱动程序开发。本文针对以上提出的系统核心问题，进行导航系统的 BSP 软件设计。

课题研究主要从 BSP 各部分组成结构，执行流程，设计思路，模型设计等方面进行了一定的研究和探讨。在研究过程中充分利用 Linux 开放源码的资源优势，进行大量的代码阅读和分析工作。着重分析并研究了 BSP 的引导代码设计，操作系统内核移植以及驱动程序框架和设计思路等，在此基础上对导航系统特定应用设计 BSP 软件。

导航系统的 BSP 软件，主要从以下几个方面着手:

(1) 采用 SAMSUNG 公司的 32 位 S3C2440 处理器作为导航系统的硬件平台。在此平台上，研究 boot loader 的工作原理，并根据嵌入式导航系统的实际硬件资源的特点，实现引导代码 boot loader 的设计，使它在目标板上电后完成硬件初始化、下载和引导内核的工作，并为应用开发人员烧写内核和文件系统设计了一个良好的人机交互界面。最后为了验证引导程序的正确性，本系统将 boot loader 编译成二进制代码，烧入 Nandflash，启动运行，对其功能进行了测试和验证。

(2) 完成了嵌入式 Linux 向嵌入式导航平台的移植，包括内核的定制与片上文件系统的设计。针对嵌入式导航系统要求文件系统占用存储空间少，系统性能好，且可写，本文提出了使用 Squanshfs 文件系统，可写的 yaffs2 文件系统和 tmpfs 文件系统组合的方法，充分利用多个文件系统的优点，满足系统的需求。并将内核和文件系统生成映像文件，通过 boot loader 的下载功能烧入 Nandflash。并对内核与文件系统的运行情况进行了测试。

(3) 通过大量驱动案例分析，研究了各类驱动的层次和接口。给出了设备驱动程序设计框架，完成导航系统的 USB 驱动、触摸屏驱动，串口驱动。

(4) 提出了若干种改善系统启动速度和占用空间的优化方法，并给出了具体的实现技术和过程，满足了嵌入式系统启动速度快，存储空间少的要求。

(5) 本文所设计的 BSP 包已在嵌入式智能交通导航设备上投入了实际的应用，并可应用于其他类似的嵌入式应用系统开发中。本文通过智能交通导航系统实例来验证所设计 BSP 的优异的性能。

1.2.2 论文结构安排

本文将分多个章节来讨论 BSP 的设计与实现，并在第七章通过一个实际的例子来检验 BSP 的正确行与可用性。下面是具体的各章任务安排：

- 第一章论述了嵌入式处理的发展现状，嵌入式操作系统的发展现状，嵌入式导航技术现状以及 BSP 技术研究现状。根据课题的研究现状提出了本文的主要的研究内容。
- 第二章给出了导航系统总体设计方案以及系统 BSP 的设计方案，分析了导航系统的硬件框架和系统工作原理，并且阐述了 BSP 各个部分的联系。
- 第三章设计 boot loader 并实现，优化了 boot loader 人机界面。
- 第四章介绍了基于 s3c2440 处理器的 Linux 操作系统移植方法，文件系统的制作方法。针对导航系统的要求，提出了将多个文件系统组合使用的方法，满足了导航系统的要求。
- 第五章介绍了 USB 设备驱动、触摸屏驱动及串口驱动的设计方法。
- 第六章从系统启动速度和文件大小两个方面优化了 BSP，给出了若干方法及其关键技术。
- 第七章给出了在本 BSP 上开发的嵌入式导航系统实例。

第二章 导航系统的设计方案

2.1 导航系统的总体设计

2.1.1 导航系统功能需求

导航系统需要提供以下几种功能：实时接收 GPS 导航卫星的定位数据，通过 GPRS 发送自己的状态信息到监控服务器，接收监控服务端的调度命令，路径规划计算，选择最优的路径，在触摸屏上显示电子地图与导航路径，用户通过触摸屏操作，提供地图管理，路径选择等多种功能的服务。

要实现上述功能，把导航系统分成 3 个部分：

（1） 导航数据接收。从全球定位系统（GPS）接收实时导航信息，通过数据通道，把数据传给 S3C2440 处理器处理。

（2） 通信模块。通过数据通道，把系统当前的状态发送到 GPRS 模块，GPRS 把数据提交给监控端，GPRS 接收监控端的指令。

（3） 数据处理模块。S3C2440 处理器根据接收到的信息通过智能导航算法作出路径规划，显示电子地图和导航路径，响应监控端的指令。

2.1.2 导航系统的硬件结构

根据系统需求，我们必须选择合适的硬件来满足系统的需要，由于本文重点介绍 BSP 的设计、实现与优化，这里只阐述硬件选型的依据，硬件电路方面本文不予叙述。

硬件主要包括以嵌入式处理器 S3C2440 为核心的数据处理存储模块、GPS 信号接收模块、GPRS 通信模块和人机交互模块四大部分。主控芯片采用三星公司的 S3C2440 处理器。通信上，通过 UART1 连接 GPS 接收机 GM-82 卫星接收模组，另通过 UART2 与 GPRS 通信模组 BENQ M23G 相连，向控制中心发送和接收数据。在人机交互上通过触摸屏实现输入输出。整个硬件框架如图 2-1 所示。

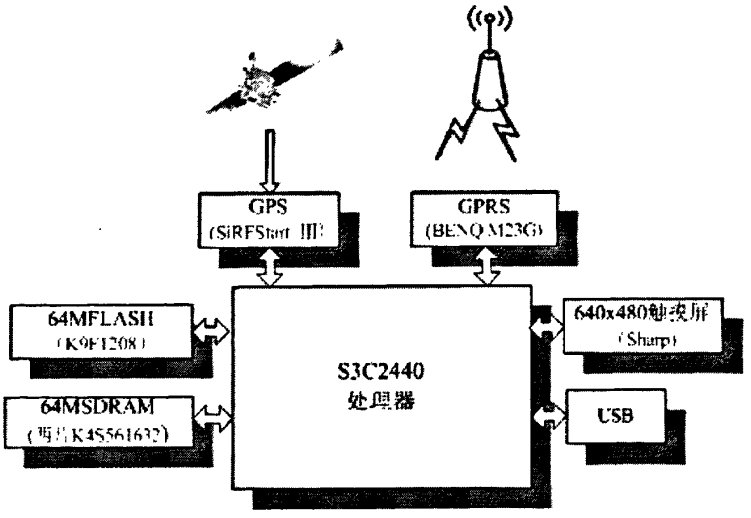


图2-1 硬件框架

Figure 2-1 Hardware framework

S3C2440性能参数介绍：

三星2440 32-bit RISC 微处理器是一款专用的以手持设备为主而设计的芯片，其特点有低功耗，高速的处理计算能力。为了减少系统的耗费，2440使用了如下组件：2440基于ARM920T内核的0.13um cmos 标准单元和存储单元复合体。 它功耗及小，简单稳定的设计非常适合对电源要求较高的产品上。它采用了新的总线构架(AMBA)，2440提供了杰出的特性。

通过提供完善的通用外围设备接口，s3c2440降低了整个系统的成本，减少了配置额外组件的需要。集成的片上系统的功能描述如下：

- 1.2v内部电压，1.8v/2.5v/3.3v 内存记忆体电压，3.3v外部i/o
- 外部存储器控制单元（控制sdram和片选）
- lcd专用控制器和专用DMA通道
- 外部4路独立DMA控制器
- 3路串口，读写64k缓存
- 2路spi口
- 2路USB host 控制器和单路设备usb控制器 1.1
- 4路定时器1路内部计数器和看门狗
- 8通道10bitADC和触摸屏幕接口
- 130路独立通用GPIO/ 24通道中断资源

GM-82 卫星接收模块性能参数介绍：

- 使用SiRF第二代高效能芯片，大大地缩小体积。
- 快速定位及追踪 12 颗卫星的能力
- 芯片内建1920 次 / 频率硬件，提高接收传送搜寻卫星讯号。
- 内建WASS/EGNOS解调器。
- 内建可充式备份锂电池，减低定位时间(TTFF)。
- 支持NMEA0183 2.2版本规格输出。
- 改良式计算理论，提高不良环境下的信号接收能力和定位准确度。

BENQ M23G模块性能参数介绍

- GSM/GPRS Class 10 三频模块。
- 符合 ETSI GSM phase 2+。
- 通讯功能：支持 GSM 语音、数据、传真、短消息及 GPRS 数据传输等。
- 软件支持标准 AT command (3GPP 27.07/27.05)。

2.1.3 导航系统的软件结构

嵌入式导航系统软件包括三个部分，应用程序，操作系统和 BSP，如图 2-2 所示。

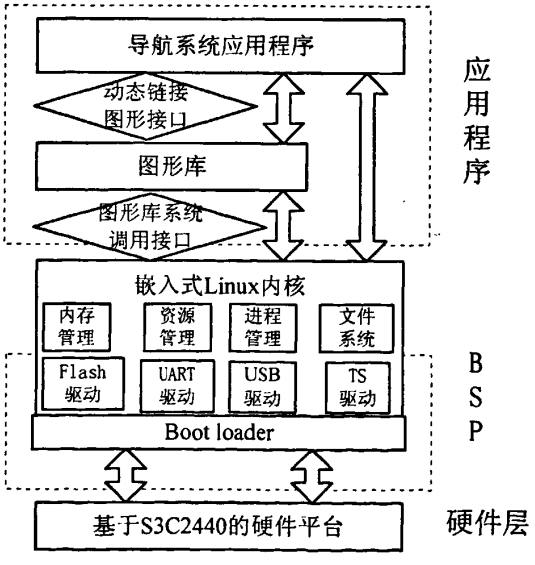


图2-2 软件框架

Figure 2-2 Software framework

应用程序属于系统的最上层，包括图形库和可执行程序，但是应用程序一般需要 linux 操作系统和 BSP 的支持，如 GPS 数据的接收和 GPRS 拨号需要串口驱动的支持，电子地图的显示和控制输入需要触摸屏驱动的支持，数据的存储需要 nandflash 驱动的支持和 USB 驱动的支持。智能导航系统路径规划算法需要操作系统 API 的支持。

操作系统提供对硬件资源的管理，提供进程调度，内存管理的功能，对于应用程序来说，它是一个黑盒子，应用程序只关心操作系统提供的 API。

BSP 为操作系统或应用程序提供最基础的支持，主要包括操作系统移植（操作系统与硬件相关的部分），设备驱动程序，bootloder。

操作系统的驱动程序，与硬件相关的部分代码，既是属于内核范畴，也属于 BSP。内核与 BSP 之间关系十分的紧密，本文把内核与硬件相关的部分都归结为 BSP。

2.2 BSP 的总体设计

2.2.1 BSP 的框架

作为嵌入式应用程序运行平台，嵌入式操作系统是最核心的软件。而作为操作系统的重要支持部件硬件相关代码也是不可缺少的，没有硬件相关代码的支持，操作系统根本无法在硬件平台上运行，也无法提供给应用程序硬件功能调用，可以说硬件相关代码是操作系统与硬件之间的一个重要桥梁。

如图 2-3 所示。BSP 设计分成 3 个模块，即 boot loader、操作系统移植与文件系统制作、驱动程序设计。

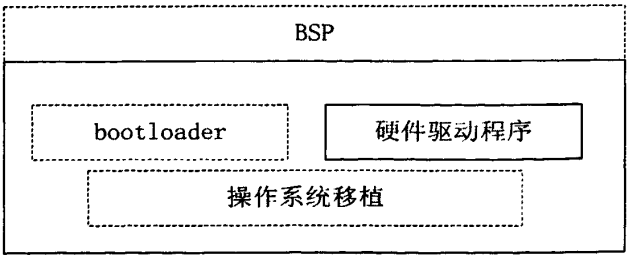


图 2-3 BSP 的组成

Figure 2-3 the composition of BSP

- 操作系统引导代码负责从外存设备加载操作系统到指定的内存空间，并完成参数传递，将 CPU 控制权转交给操作系统的工作。由于不同操作系统一般都有不同的加载地址和加载参数，因此操作系统引导代码往往也随操作系统的不同而有所变化。
- 操作系统移植和文件系统制作。操作系统移植主要是实现与硬件相关部分的代码，负责完成对硬件的探测、检查、登记、初始化等工作。在操作系统源代码的 arch 文件夹下，操作系统的移植主要是对与硬件相关部分的修改。文件系统制作包括文件系统内容的准备以及利用文件系统工具制作文件系统映像。
- 硬件设备驱动程序是与操作系统相关的程序代码和程序框架。它既包括了硬件各功能的功能函数，也包括了与操作系统相关的驱动程序框架结构。由于硬件设备的多样性，决定了驱动程序具有很大的代码开发工作量[18]。

2.2.2 BSP 的优化

BSP 包括 boot loader、Linux 内核移植、设备驱动程序三个部分。

本文在 boot loader 设计过程中采用优化设计方法，改变原来命令行操作方式，设计了 boot loader 的人机交互界面。使应用程序开发人员不需要了解 FLASH 烧写命令，不需要了解 Nandflash 的分区结构，也不需要了解 boot loader 的操作命令，按照提示选择 0、1、2、3 就可以完成内核与文件系统的烧写。

本文还采用了优化的内核移植方法，提出了采用组合的文件系统提高系统存储空间利用率以及 mount 速度，并且专门研究了改善系统启动速度和减小文件大小的优化方法，采用了预设 lpj，并行加载模块，关闭串口打印信息，用 uclibc 库替换 glibc 等系统方法，提高系统的启动速度，减少了文件的大小。使系统成本最低，性能最佳。

2.3 本章小结

本章讨论了系统的总体设计，根据系统的要求选择合适的硬件平台，搭建软、硬件平台的框架，并结合课题要求，给出了 BSP 的总体设计方案，详细讨论了 BSP 包的组成，并提出了 BSP 系统的优化设计思路与方法。

第三章 Boot loader 的设计与实现

3.1 Boot loader 原理

3.1.1 Boot loader 概念

Boot loader (引导加载程序) 主要完成初始化硬件设备，建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为调用操作系统内核准备好正确的环境[19]。目前流行的可以引导加载Linux 的Boot loader 包括: 支持x86 体系结构的 LILO、GRUB、ROLO、Loadlin、Etherboot、LinuxBIOS；支持 ARM 体系结构的 Compaq 的 bootldr、Blob、U-Boot、vivi、RedBoot；支持 MIPS 体系结构的 PMON 和支持 m68k 体系结构的 sh-boot[20]。

通常，Boot loader 是非常依赖于硬件而实现的，特别是在嵌入式世界。因此，在嵌入式世界里建立一个通用的 Boot loader 几乎是不可能的。所以我们必须根据特定的环境开发自己的 Boot loader。本文针对基于 ARM 的导航系统的特定硬件环境和系统要求，设计与实现具有针对性的 Boot loader。

3.1.2 Boot loader 的安装媒介

系统加电或复位后，所有的 CPU 通常都从某个由 CPU 制造商预先安排的地址上取指令。比如，基于 ARM7TDMI core 的 CPU 在复位时通常都从地址 0x00000000 取它的第一条指令。而基于 CPU 构建的嵌入式系统通常都有某种类型的固态存储设备（比如：ROM、EEPROM 或 FLASH 等）被映射到这个预先安排的地址上。因此在系统加电后，CPU 将首先执行 Boot loader 程序[21]。

图 3-1 就是一个同时装有 Boot loader、内核的启动参数、内核映像和根文件系统映像的固态存储设备的典型空间分配结构图。

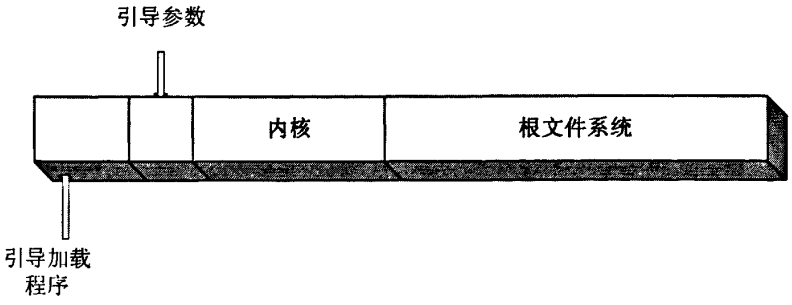


图 3-1 固态存储设备的典型空间分配结构

Figure 3-1 Solid-state storage device structure of a typical spatial distribution

3.1.3 Boot loader 的操作模式

Boot loader 都包含两种不同的操作模式："启动加载"模式和"下载"模式，这种区别仅对于开发人员才有意义。但从最终用户的角度看，Boot loader的作用就是用来加载操作系统，而并不存在所谓的启动加载模式与下载工作模式的区别。

启动加载（Boot loading）模式：这种模式也称为"自主"（Autonomous）模式。即Boot loader 从目标机上的某个固态存储设备上将操作系统加载到RAM中运行，整个过程并没有用户的介入。这种模式是Boot loader的正常工作模式，因此在嵌入式产品发布的时侯，Boot loader显然必须工作在这种模式下。

下载（Downloading）模式：在这种模式下，目标机上的 Boot loader 将通过串口连接或网络连接等通信手段从主机（Host）下载文件，比如：下载内核映像和根文件系统映像等。从主机下载的文件通常首先被 Boot loader 保存到目标机的 RAM 中，然后再被 Boot loader 写到目标机上的 FLASH 类固态存储设备中。Boot loader 的这种模式通常在第一次安装内核与根文件系统时被使用。此外，以后的系统更新也会使用 Boot loader 的这种工作模式。工作于这种模式下的 Boot loader 通常都会向它的终端用户提供一个简单的命令行接口。

3.2 Boot loader 实现

3.2.1 boot loader 的开发测试环境

建立完善的开发与测试环境。基于 arm 平台，现在有非常成熟的集成开发环境[23]。如：ADS、RealView。本系统采用如下环境：

开发软件：CodeWarrior for ADS1.2 集成开发环境

调试软件：AXD、ARM Multi-ICE

调试硬件工具：Multi-ICE 仿真器、JTAG 线

串口工具：DNW

烧写软件：sjf2440

3.2.2 本系统 boot loader 实现的关键点

- 下载方式：

由于 Boot loader 的实现依赖于 CPU 的体系结构和外部设备，导航系统通过 GPRS 网络通信，没有以太网接口，所以我们在设计 boot loader 时必须考虑文件映像下载方式。根据本系统的硬件资源，可以考虑使用串口方式和 USB 口方式下载文件，但串口下载速度非常缓慢，下载 boot loader 这样的小文件用户可以接受，如果下载 zImage 和文件系统这样大文件，用户或者应用程序开发人员等待下载的时间太长，是不可接受的。所以必须使用其他下载方式，比如 USB 口下载。我们系统板提供了 USB device 端口，在引导程序中我们加入 USB 驱动，利用 USB 下载文件，加快下载速度。

- Nandflash 启动

S3C2440 集成了一个 Nandflash 控制器和一个容量为 4KB 的 SDRAM 缓存（称作 Steppingstone），能够在系统启动时将 Nandflash 的前 4KB 内容拷贝到 Steppingstone 中，然后将 Steppingstone 映射到 0x0 的地址后，从 0x0 开始启动运行。因此启动代码前 4KB 的内容，必须具有拷贝自身到 RAM 的功能。通过跳转指令跳转到 RAM 中运行。然后，加载操作系统内核，传递内核启动参数，完成加载功能。

- 人性化的人机界面

在开发的过程中，大多数开发人员都忽视了 boot loader 的人机界面，因为传统的 boot loader 都是采用命令行的方式，如 u-boot。给应用程序开发人员和系统移植人员带来了操作上的不便。本系统提出了 boot loader 的人机界面的问题，采用简单的字符界面，应用程序开发人员只要通过选择 0，1，2，... 就可以完成 boot loader 的更新，操作系统和文件系统的烧写，给后续的开发工作带来了便利。

3.2.3 boot loader 程序实现

我们将 Boot loader 分为 stage1 和 stage2 两大部分。依赖于 CPU 体系结构的代码，比如设备初始化代码等，通常都放在 stage1 中，而且通常都用汇编语言来实现，以达到短小精悍的目的。而 stage2 则通常用 C 语言来实现，这样可以实现更复杂的功能，而且代码会具有更好的可读性和可移植性。

3.2.3.1 Boot loader 的 stage1

- 基本的硬件初始化

这是 Boot loader 一开始就执行的操作，其目的是为 stage2 的执行以及随后的 kernel 的执行准备好一些基本的硬件环境。它通常包括以下步骤：

1. 屏蔽所有的中断。为中断提供服务通常是 OS 设备驱动程序的责任，因此在 Boot loader 的执行全过程中可以不必响应任何中断。中断屏蔽通过写 CPU 的中断屏蔽寄存器来完成。

```
ldr  r0,=INTMSK
ldr  r1,=0xffffffff   ;all interrupt disable
str  r1,[r0]
ldr  r0,=INTSUBMSK
ldr  r1,=0x3ff        ;all sub interrupt disable
str  r1,[r0]
```

2. 设置 CPU 的速度和时钟频率。(cpu 频率为 400M,系统时钟是 100M，慢速设备时钟为 50M)，根据 S3C2440 数据手册，先确定 FLCK、HCLK、PCLK 之间的比例为 1：4：8，得到 CLKDIVN 为 5 [24]。把值写入 CLKDIVN 寄存器，完成设定。

```
ldr   r0, =CLKDIVN
mov r1, #5
str   r1, [r0]
```

3. RAM 初始化。设置系统的内存控制器的功能寄存器以及各内存库控制寄存器等。

```
    ;Set memory control registers
    adr  r0, SMRDATA   ;can't use ldr r0, =xxxx important!!!
    ldr  r1, =BWSCON   ;BWSCON Address
    add r2, r0, #52    ;End address of SMRDATA
0
    ldr  r3, [r0], #4
    str  r3, [r1], #4
    cmpr2, r0
```

4. 关闭 CPU 内部指令 / 数据 cache。

5. 设置堆栈指针 sp。

堆栈指针的设置是为了执行 C 语言代码作好准备。

- 检查是否从nandflash启动

S3C2440处理器支持从Norflash启动。同时支持从Nandflash启动。这里从Nandflash启动。

- 拷贝 boot loader 到 RAM 中

拷贝时要确定两点：(1) boo tloader的可执行映象在固态存储设备的存放起始地址和终止地址；(2) RAM 空间的起始地址。S3C2440集成了Nandflash控制器，通过操作寄存器来完成数据的拷贝。

- 跳转到RAM运行

InitRam之后，跳转到RAM运行。

- 跳转到 stage2 的C入口点

在上述一切都就绪后，通过bl main指令，就可以跳转到 Boot Loader 的 stage2 去执行了。

3.2.3.2 Boot loader 的 stage2

正如前面所说，stage2 的代码通常用C语言来实现，以便于实现更复杂的功能和取得更好的代码可读性和可移植性。本系统Stage2的设计步骤如下：

- 跳转main()函数

如何跳转到main()函数呢？这里运用了一种巧妙的方法是利用trampoline(弹簧床)的概念。即用汇编语言写一段trampoline小程序，并将这段trampoline小程序来作为stage2可执行映象的执行入口点。然后我们可以在trampoline汇编小程序中用 CPU 跳转指令跳入main()函数中去执行；而当main()函数返回时，CPU执行路径显然再次回到我们的trampoline程序。简而言之，这种方法的思想就是：用这段

trampoline小程序来作为main()函数的外部包裹(external wrapper)。

```
.text
_trampoline:
bl main
/* if main ever returns we just call it again */
b _trampoline
```

可以看出，当main()函数返回后，我们又用一条跳转指令重新执行 trampoline 程序——当然也就重新执行main()函数。

- 初始化本阶段要使用到的硬件设备

这通常包括：（1）初始化一个串口，终端用户进行 I/O 输出信息；（2）初始化计时器；（3）检测系统的内存映射(memory map）。

- 打印boot loader的人机交互界面

通过串口在超级终端中打印出boot loader启动人机界面。如图3-2所示

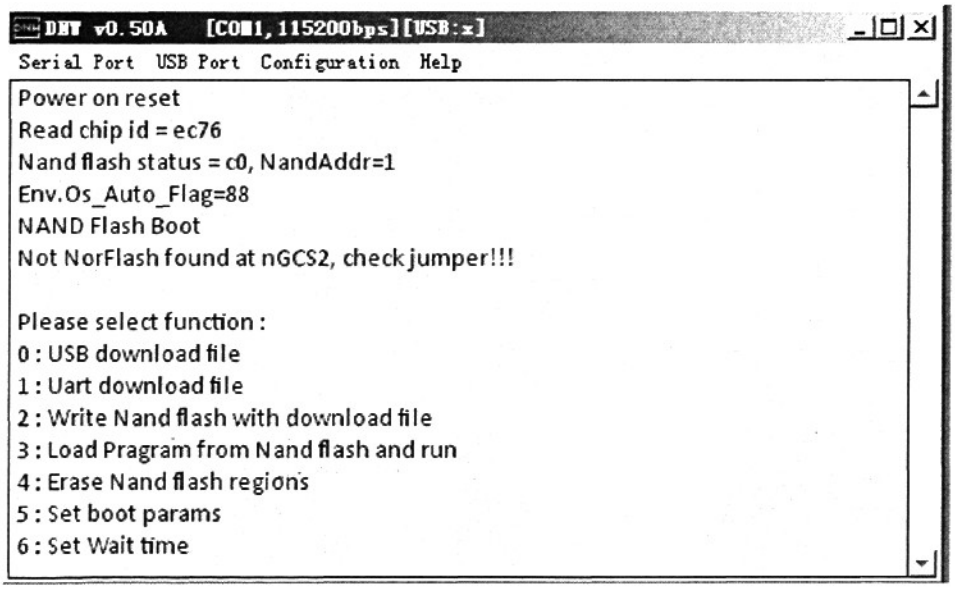


图3-2 boot loader下人机交互界面

Figure 3-2 boot loader man-machine interface

- 提供支持函数

为人机界面提供支持，需要Nandflash驱动、USB驱动、串口驱动、参数设置函数Nandflash在本系统分为5个分区：boot loader、kernel、squanshfs、yaffs2、maps。串口波特率设置为115200。

- 加载内核映像和根文件系统映像
- 设置内核的启动参数

在将内核映像和根文件系统映像拷贝到 RAM 空间中后，就可以准备启动 Linux 内核了。但是在调用内核之前，还需作一步准备工作：设置 Linux 内核的启动参数如下：

"root=/dev/mtdblock2 load_ramdisk=0 init=/Linuxrc console=ttySAC0 display=sharp640 mem=64M devfs=mount"

- 调用内核

Boot loader调用Linux内核的方法是直接跳转到内核的第一条指令处，在跳转时，下列条件要满足：

CPU 寄存器的设置：R0＝0;R1＝机器类型 ID;关于 Machine Type Number，可以参见 Linux/arch/arm/tools/mach-types。 R2＝启动参数标记列表在 RAM 中起始基地址；

CPU 模式：必须禁止中断（IRQs和FIQs）；CPU 必须 SVC 模式；

Cache 和 MMU 的设置： MMU 必须关闭；指令 Cache 可以打开也可以关闭；数据 Cache 必须关闭；

3.2.4 boot loader 测试

本系统已成功实现 boot loader，Boot loader 完成了 2 个功能：一是下载功能，二是引导功能。引导功能测试我们在 Linux 内核移植这一章详细叙述。

首先要通过 jtag 把 boot loader 烧写入内核，烧写过程见图 3-3，根据提示选择。

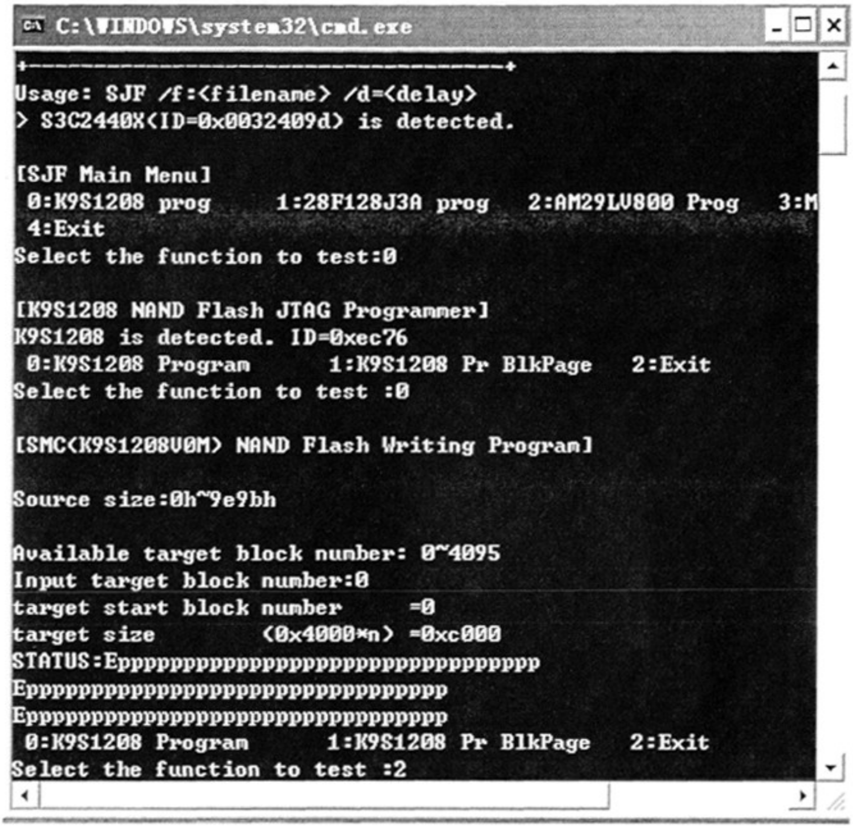


图 3-3 jtag 烧写

Figure 3-3 jtag program

下载功能测试：打开 dnw 应用程序，插好 USB 线，启动电源，进入 boot loader 启动界面，如图 3-2 所示，选择 0，进入 USB 下载界面，如图 3-4 所示，选择要下载的文件传入 RAM,下载完毕后显示 Download O.K，询问是否要运行，选择否，（当测试程序时，而不想写入 Nandflash 时，可以选择是）。接着选择 2（Write Nandflash with download file），用刚才下载到内存的文件写入到 Nandflash，再选择 Nandflash 分区，就是选择要写入的哪个地方，假如烧写内核就选择 1 kernel 分区。Bootloader 最后会询问是否确定要写入，选择是。烧写成功后将打印"Program nand flash partirion success"信息，表明下载成功。

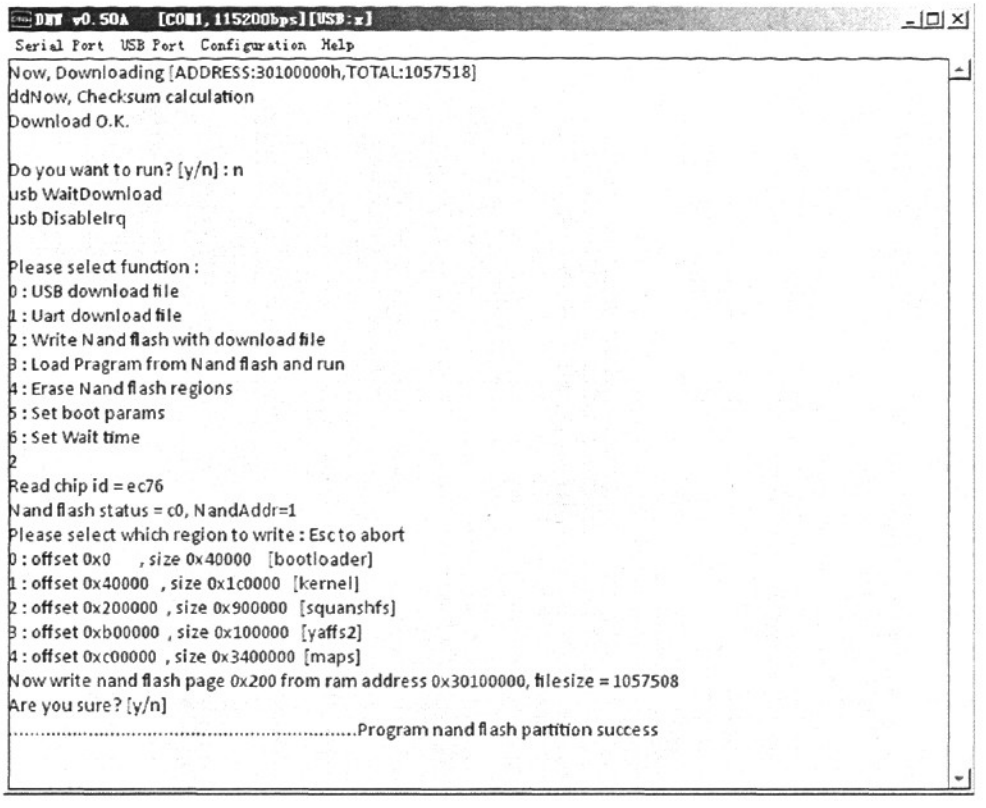


图 3-4 下载功能测试

Figure 3-4 Download test

3.3 本章小结

本章介绍了 Linux 引导程序 boot loader 的概况，boot loader 引导程序的启动原理，根据引导系统的启动过程。重点介绍了针对导航系统的 boot loader 的设计方法与关键技术，并对传统的 boot loader 引导人机界面进行了优化设计与实现。

第四章 操作系统移植与文件系统制作

4.1 嵌入式 Linux 的移植

4.1.1 Linux 操作系统的优势与特点

与其他的商业操作系统相比，Linux有如下优势：

程序代码的质量与可靠性

质量与可靠度是对程序代码有多少信心的主观度量。优质的代码应该具备以下特性：

- 模块化与结构化
- 容易修改
- 可扩充
- 可配置

可靠的程序代码应该具备的特性：

- 可预测
- 从错误中恢复的能力
- 长期运行的能力

Linux 内核及系统上大部分项目的程序代码库都符合以上对质量和可靠度所做的描述。

程序代码的可用性

Linux可以毫无限制地随意取用Linux的源码及所有生成工具，最重要的Linux组件，包括内核本身，都是在GNU通用公共许可证的保护下发行的，因此取用这些组件的源代码是每个人的权利。

对硬件的支持

广泛的硬件支持，意味着 Linux 支持各种不同类型的硬件平台与设备。因为有大量的驱动程序是靠 Linux 社群自己维护的，所以你大可放心的使用这些硬件组件，而不必担心有朝一日厂商会停止该生产线。

通信协议与软件标准

Linux 提供广泛的通信协议及软件标准支持。这让我们可以轻易地将 Linux 整合进既有架构，以及将过去的软件移植到 Linux 上。你可以轻易地将 Linux 系统整合进既有的 Windows 网络。

成本

使用 Linux 所有开发工具和操作系统组件都可以免费获得，而且它发行时的许可证就是为了避免有人对内核组件收取任何版税。与传统嵌入式软件的成本模型比起来，这已经是相当低的成本了 [21]。

4.1.2 Linux 的组成

嵌入式Linux内核指的是一个提供硬件抽象层、磁盘及文件系统控制、多任务等功能的系统软件[25]。

Linux内核主要由五个子系统组成：进程调度，内存管理，虚拟文件系统，网络接口，进程间通信。

- 进程调度(SCHED)

控制进程对CPU的访问。当需要选择下一个进程运行时，由调度程序选择最值得运行的进程。可运行进程实际上是指等待CPU资源的进程，如果某个进程在等待其它资源，则该进程是不可运行进程。Linux使用了比较简单的基于优先级的进程调度算法选择新的进程。

- 内存管理(MM)

允许多个进程安全的共享主内存区域。Linux的内存管理支持虚拟内存，即在计算机中运行的程序，其代码，数据，堆栈的总量可以超过实际内存的大小。操作系统只是把当前使用的程序块保留在内存中，其余的程序块则保留在磁盘中。必要时，操作系统负责在磁盘和内存间交换程序块。内存管理从逻辑上分为硬件无关部分和硬件有关部分。硬件无关部分提供了进程的映射和逻辑内存的对换；硬件相关的部分为内存管理硬件提供了虚拟接口。

- 虚拟文件系统(Virual FileSystem，VFS)

隐藏了各种硬件的具体细节，为所有的设备提供了统一的接口，VFS提供了多达数十种不同的文件系统。虚拟文件系统可以分为逻辑文件系统和设备驱动程序。逻辑文件系统指Linux所支持的文件系统，如ext2, FAT，等，设备驱动程序指

为每一种硬件控制器所编写的设备驱动程序模块。

- 网络接口(NET)

提供了对各种网络标准的存取和各种网络硬件的支持。网络接口可分为网络协议和网络驱动程序。网络协议部分负责实现每一种可能的网络传输协议。网络设备驱动程序负责与硬件设备通讯，每一种可能的硬件设备都有相应的设备驱动程序[27]。

- 进程间通讯(IPC)

支持进程间各种通信机制

处于中心位置的进程调度，所有其它的子系统都依赖它，因为每个子系统都需要挂起或恢复进程。一般情况下，当一个进程等待硬件操作完成时，它被挂起;当操作真正完成时，进程被恢复执行。例如，当一个进程通过网络发送一条消息时，网络接口需要挂起发送进程，直到硬件成功地完成消息的发送，当消息被成功地发送出去以后，网络接口给进程返回一个代码，表示操作的成功或失败。其他子系统以相似的理由依赖于进程调度。

各个子系统之间的依赖关系如图4-1所示：

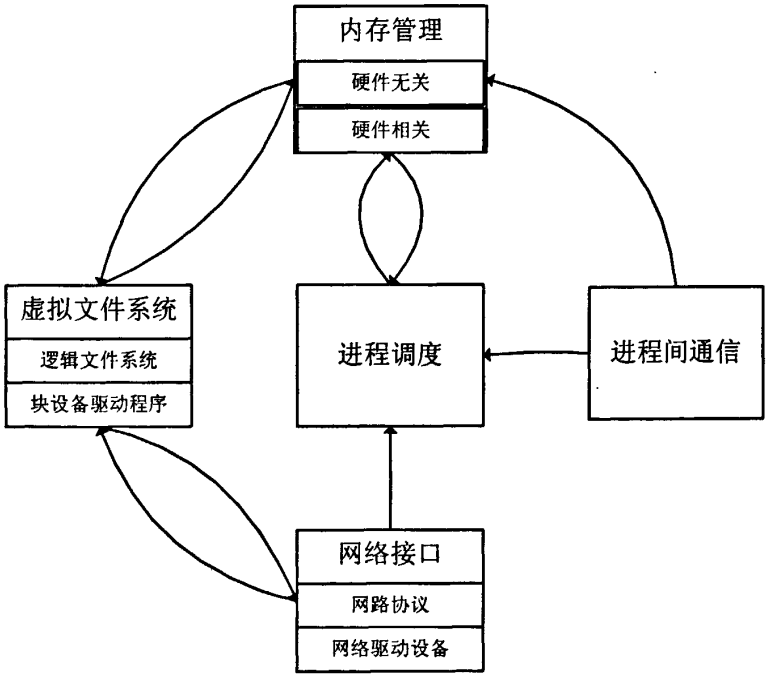


图4-1 Linux各子系统之间的关系

Figure 4-1 the relationship between the various subsystems in Linux

- 进程调度与内存管理之间的关系

这两个子系统互相依赖。在多道程序环境下，程序要运行必须为之创建进程而创建进程的第一件事情，就是将程序和数据装入内存。

- 进程间通信与内存管理的关系

进程间通信子系统要依赖内存管理支持共享内存通信机制，这种机制允许两个进程除了拥有自己的私有空间，还可以存取共同的内存区域。

- 虚拟文件系统与网络接口之间的关系

虚拟文件系统利用网络接口支持网络文件系统(NFS)，也利用内存管理支持RAMDISK设备。

- 内存管理与虚拟文件系统之间的关系

内存管理利用虚拟文件系统支持交换，交换进程(swapd) 定期由调度程序调度，这也是内存管理依赖于进程调度的唯一原因。当一个进程存取的内存映射被换出时，内存管理向文件系统发出请求，同时，挂起当前正在运行的进程。除了这些依赖关系外，内核中的所有子系统还要依赖于一些共同的资源。这些资源包括所有子系统都用到的过程。例如:分配和释放内存空间的过程，打印警告或错误信息的过程，还有系统的调试例程等[26]。

4.1.3 嵌入式操作系统的移植

4.1.3.1 内核移植目标

Linux内核的剪裁和移植目标是面向具体应用对象的硬件平台的，针对嵌入式导航系统的实际需求，完成嵌入式Linux的剪裁和移植，需要满足如下条件:

将内核配置成一个最小的系统，保留必要的设备驱动，必要的功能模块，特定的机制，特定的协议，去掉不用的功能模块。例如特定的USB模块、串口模块和触摸屏模块、电源管理机制等，这些需要保留， 而像SD卡模块就可以去掉。

4.1.3.2 编译环境的建立

移植嵌入式Linux和编译应用程序都需要在主机上建立交叉编译环境

建立交叉编译环境，是为了在主机上构建一个编译内核和应用程序的环境，使得执行代码可以在目标板上运行，目标板的硬件并不一定可以执行在X86系统上可以执行的二进制代码。我们需要将各种二进制工具程序集成进工具链，其中

包括如ld、gas、ar、C/C++(gcc/g++)编译器以及C库（glibc）。工具链有两种方法获得：自己制作工具链或从网站上下载成熟的工具链。如果是自己建立工具链可以登陆网站下载工具链的各个组件：Binutils包位于binutils目录、gcc包位于gcc目录，而glibc包则与glibc-Linuxthreads包一起放在 glibc目录[28]。根据文档选择合适的版本进行组合编译。如果使用已经成熟的交叉编译工具链，其下载网站地址：ftp://ftp.arm.Linux.org.uk/pub/armLinux/toolchain/

本系统目标板采用的是ARM S3C2440的处理器平台，所以它的交叉编译环境具体如下:

1) 主机环境

主机是Redhat Enterprise Linux AS V4的Linux环境，CPU是Intel的pentium(R).Dual T2390 内存1G 。

2) 针对目标板的交叉编译器

采用 arm-Linux-gcc作为GNU的c交叉编译器

3) 针对目标板的应用程序链接库

由于编译内核，不用到c库，因此不具体设定交叉编译环境下的应用程序链接库

4.1.3.3 操作系统内核的选择

Linux社区是一个更新很快的社区，他们不断努力提供对新的硬件的支持，修订原来的一些bugs，已经在时间响应上做出更好的改善，使得Linux 的release版本可以在6个月甚至更少的时间得到更新。在Linux 社区中，不同的组织维护着不同的Linux分支，选择Linux版本，需要考虑内核的稳定性，以及内核版本是否满足目标平台要求，所以在选择的时候不仅要确定Linux内核，同时要关注新发布的补丁，作为内核的补充，满足目标架构[29]。

系统采用Linux2.6.14内核，它可以满足嵌入式导航系统的应用要求，并且由于这个内核已经经过了很长的时间的维护，目前是非常稳定的版本，非常适应系统需要较强的稳定性的要求。

4.1.3.4 操作系统移植原理

要进行操作系统的移植，首先要修改Linux硬件相关的代码。Linux的文件组

织形式非常清晰。Linux内核架构可以分成与硬件相关的部分，以及与硬件独立的部分。与硬件相关的部分会在最先执行，它们负责配置硬件寄存器，存储空间映射，设置中断，使能CACHE，配置页大小，进行其余硬件初始化，然后开始硬件独立部分的代码执行。Linux试图在硬件无关和硬件相关的源码之间维持一个清晰的划分[30]。

Linux内核可以视为由五个功能部分组成：进程调度（包括调度和通信），内存管理，虚拟文件系统，网络接口，进程间通信。上面所讲的五个部分的顺序不是随便排的，从前到后分别代表着它们与硬件设备的相关程度。越靠前越高，后面的两个虚拟文件系统和网络则几乎与平台无关，它们由设备管理中所支持的驱动程序提供底层支持。因此，在做系统移植的时候，需要改动的就是进程管理、内存管理和设备管理中被独立出来的那部分即硬件相关部分的代码。在Linux代码树下，这部分代码全部在arch目录下[30]。

进程管理底层代码：从硬件系统的角度来看，进程管理就是CPU的管理。在不同的硬件平台上，这有很大的不同。CPU中用的寄存器结构不同，上下文切换的方式、现场的保存和恢复、栈的处理都不同，这些内容主要由CPU开发手册所描述。通常来说，CPU的所有功能和状态对于Linux不一定有意义。实现时，需要在最小的开发代价和最好的系统性能之间加以权衡。

时钟、中断等板上设备支持代码：在同一种CPU的平台上，也会存在不同的板上外设，异种CPU平台上更是如此。不同的系统组态需要不同的初始化代码。针对S3C2440处理器，需要做如下的改动：

- 完成外设的虚拟地址空间映射

根据外设的硬件连接，设置外设虚拟寻址空间。

- 完成中断设置

配置中断脚，分配中断号，并且指定中断的边沿信号。

- 设置外设的片选

设置外设的片选信号，确定CPU对外设的读写时序。

- 设置USB控制器驱动

按照硬件连接，设置片选信号，注册中断号和中断引脚，设置USB设备的虚拟地址空间。

所有与平台相关的内存管理代码全部在在arch/arm/mm/下的内存管理部分。这

部分代码完成内存的初始化和各种与内存管理相关的数据结构的建立。Linux使用了基于页式管理的虚拟存储技术，而CPU发展的趋势要使实现内存管理的功能单元统统被集成到CPU中。因此内存管理成为一个与CPU十分相关的工作。同时内存管理的效率也是最影响系统性能的因素之一。

内存可以说是计算机系统中最频繁访问的设备，如果每次内存访问时多占用一个时钟周期，那就有可能将系统性能降低到不可忍受。在Linux系统里，不同平台上的内存管理代码的差异程度是令人吃惊的，可以说是差异最大的。不同的CPU有不同的内存管理方式，同一种CPU还会有不同的内存管理模式。有些平台上甚至只是用最保守的内存管理模式。除此之外，还有一些代码需要考虑，但相对来说次要一些[31]。

4.1.3.5 内核移植过程

由于Linux内核具备可移植性的特点，并且已经支持了很多种目标板，这样，用户很容易找到满足需求的硬件平台，参考内核已经支持的目标板来进行移植工作。Linux2.6.14已经支持S3C2410处理器的多种硬件板，众所周知，S3C2440与S3C2410是同一厂商的系统结构相同的产品，他们有很多相似之处，我们在S3C2440处理器移植内核的过程中，可以参考SMDK2410参考板来移植系统的内核。所以，虚拟空间、中断等方面都不需要进行修改。我们只要对其进行一些必要的修改。

本文针对嵌入式导航系统的硬件平台进行内核的移植，具体移植过程如下：

一、修改底层与硬件相关的代码

1、对照S3C2440数据手册，修改cpu的频率为400MHZ，创建目标板文件。

2、设置flash分区，需指明分区信息，指定启动时初始化，禁止Flash ECC校验，此处一共要修改3个文件，分别是在arch/arm/mach-s3c2440目录下的devs.c和mach-smdk2440.c 在drivers/mtd/nand目录下的s3c2440.c

1)、指明分区信息

在 arch/arm/mach-s3c2440/devs.c 文件添加的内容包括：

a) 添加包含头文件。

```
#include <Linux/mtd/partitions.h>
#include <Linux/mtd/nand.h>
#include <asm/arch/nand.h>
```

b) 建立 Nand Flash 分区表

```
static struct mtd_partition partition_info[] ={
    { /* 256kB */
        name: "bootloader",
        size: 0x00040000,
        offset: 0x0,
    },{ /*1.75MB */
        name: "kernel",
        size: 0x001C0000,
        offset: 0x00040000,
    }, { /* 9MB */
        name: "squanshfs",
        size: 0x00900000,
        offset: 0x00200000,
    }, { /* 1MB */
        name: "yaffs2",
        size: 0x00100000,
        offset: 0x00b000000,
    }, { /* 52MB */
        name: "maps",
        size: 0x03400000,
        offset: 0x00c000000,
    }
};
```

目标板 Nandflash 分 5 个区，分别存放 boot loader, kernel, squanshfs, yaffs2, maps。

c) 加入 Nand Flash 分区

```
struct s3c2440_nand_set nandset ={
nr_partitions: 5,          /* 指明 partition_info 中定义的分区数目 */
partitions: partition_info, /* 分区信息表*/
};
```

d) 建立 Nand Flash 芯片支持

```
struct s3c2440_platform_nand superlpplatform={
    tacls:0,
    twrph0:30,
    twrph1:0,
    sets: &nandset,
    nr_sets: 1,
};
```

e) 加入 Nand Flash 芯片支持到 Nand Flash 驱动

另外，还要修改此文件中的 s3c_device_nand 结构体变量，添加对 dev 成员的赋值

```
struct platform_device s3c_device_nand = {
        .name = "s3c2440-nand",   /* Device name */
        .id = 1,   /* Device ID */
        .num_resources = ARRAY_SIZE(s3c_nand_resource),
        .resource = s3c_nand_resource, /* Nand Flash Controller Registers */
```

```
        /* Add the Nand Flash device */
        .dev = {
            .platform_data = &superlpplatform
        }
    };
```

2)、指定启动时初始化

kernel 启动时依据我们对分区的设置进行初始配置。在 arch/arm/mach-s3c2440/mach-smdk2440.c 文件中修改 smdk2440_devices[].指明初始化时包括我们在前面所设置的 flash 分区信息

```
static struct platform_device *smdk2410_devices[] __initdata = {
    ..............
    &s3c_device_iis,
    /* 添加如下语句即可 */
    &s3c_device_nand,
};
```

3、支持启动时挂载 devfs

为了我们的内核支持 devfs 以及在启动时并在/sbin/init 运行之前能自动挂载/dev 为 devfs 文件系统，修改 fs/Kconfig 文件，找到 menu “Pseudo filesystems”，添加如下语句：

```
config DEVFS_FS
    bool "/dev file system support (OBSOLETE)"
    default y
config DEVFS_MOUNT
    bool "Automatically mount at boot"
    default y
    depends on DEVFS_FS
```

底层跟硬件相关的代码修改完毕

二、编译内核

编译内核必须在交叉编译环境下，并且修改环境变量$(PATH)，使其包括交叉编译器所在路径。要正确使用make命令编译内核必须设置正确的Makefile文件。设置$(ARCH)为当前处理器的类型。设置$(CROSS-COMPILER)为当前交叉编译器类型。

使用Make menuconfig命令，配置内核选项[32]。

- Code maturity level options

在这部分 ，允许内核的一些试验选项(experimental options)。有时，这些选项是必需的，比如，需要支持新的显卡。但是，多数情况下，这些试验选项会导致内核不稳定，应慎重考虑是否使用。

- Loadable module support

可加载模块是指内核代码(kernel code)的一些片断，比如驱动程序译内核的时候它们也被单独编译。因此，这些代码不是内核的一部分，你需要它的时候，它可以被加载并使用。

- System type

  选择处理器(Processor)的类型为S3C2440

- General setup

  制定特定的内核常规选项。"System V IPC"允许程序通信和同步，"BSD process accounting"保持诸如进程结束时产生的错误代码的东西，"Sysctl support"允许程序修改某些内核选项而不需要重新编译内核或者重新启动计算机。设置系统启动的命令行参数，需要指定系统启动的方式，根文件系统的挂载方式和存储位置。

- Memory Technology Devices

  系统必须加载MTD系统，用于支持存储设备。

- ATA/IDE/MFM/RLL support

  USB设备需要被仿真为SCSI才能被访问，要仿真SCSI,需要打开选项SCSI emulation support

- Character devices

  打开选项 Samsung s3c2440 serial port support 和 support for conole on s3c2440 serial port支持串口以及串口终端。

  打开选项s3c2440 touchscreen drivers支持触摸屏。

- File system

  系统需要支持两种文件系统格式:squashfs文件系统作为根文件系统，以及yaffs2文件系统作为NAND Flash的文件系统。需要打补丁，在移植文件系统一节有详细介绍。

- USB support

  对USB控制器驱动的支持，配置USB设备为Mass-storage device.

内核可以被编译成多种格式的映象格式，vmLinux和zImage是普遍选择两种内核镜像格式，vmLinux与zImage的区别在于，vmLinux是没有压缩的可执行镜像，

而zImage是压缩的镜像，它们存储着相似的内容，系统选择zImag。因为可以节省存储空间，同时可以降低成本[33]。

4.1.4 内核移植的测试结果及分析

本系统已成功实现了嵌入式导航系统的Linux内核移植并进行了测试，其测试包括两个方面，其一，boot loader引导功能的测试，它是第三章的遗留下来，其二，Linux内核启动测试。

把内核映像烧入flash，启动目标板。测试结果如图4-2，boot loader引导成功，系统正常启动。

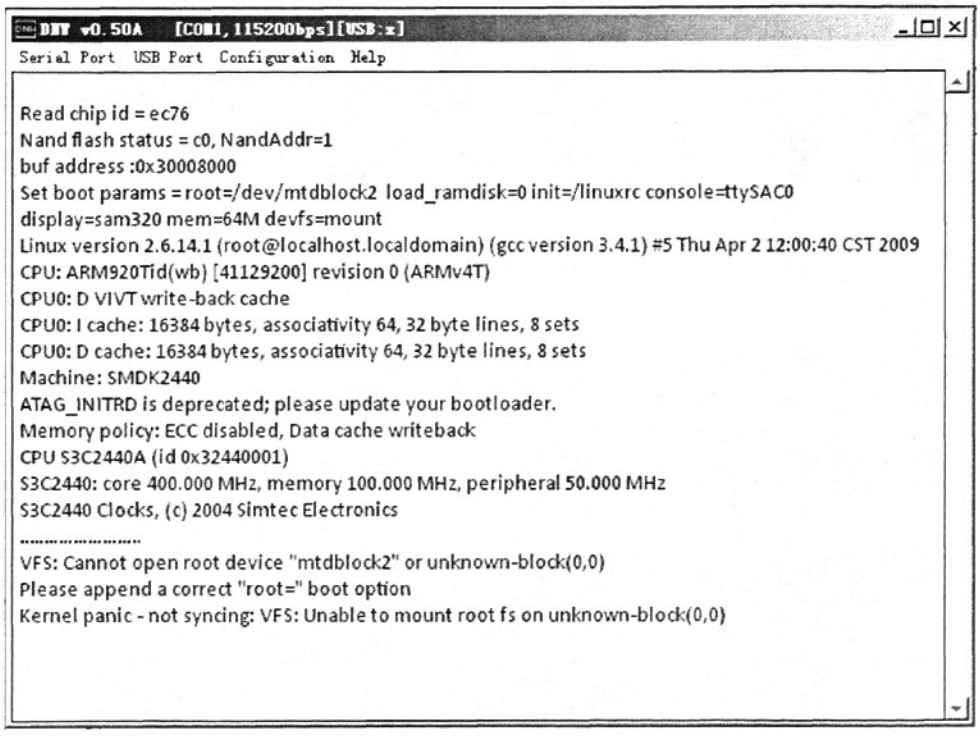


图4-2 Linux启动信息

Figure 4-2 Linux startup information

系统启动信息中出现了VFS: Cannot open root device“mtdblock2” or unknown-block(0,0)的错误信息，但在这里不是错误，是没有烧写文件系统所致，下一节中我们将讨论文件系统的移植。

4.2 文件系统的移植

4.2.1 各种文件系统的比较

要为特定的应用选择最佳的文件系统或最佳的文件系统组合，我们需要了解文件系统的特性。如何评价一个文件系统，本文将从以下几个方面进行评估

（1） 是否可写

这个文件系统是否可以被写入，如要保存数据，文件系统则要求可写。

（2） 具有永久性

重新引导之后这个文件系统时候可以保存修改过的内容。

（3） 具有断电可靠性

经变动的文件系统是否可以再断电之后恢复过来。

（4） 经过压缩

安装的文件系统，其内容是否经过压缩。

（5） 存在于RAM中

文件系统的内容在被安装之前会先从存储设备取出并放到RAM中。

表4-1 文件系统特性比较

Table 4-1 Comparison of File System Characteristics

文件系统	可被写入	具有永久性	具有断电可靠性	经过压缩	存在于RAM中
CRAMFS	否	不适用	不适用	是	否
SQUASHFS	否	不适用	不适用	是	否
JFFS2	是	是	是	是	否
JFFS	是	是	是	否	否
RAMDISK	否	否	否	是	是
YAFFS2	是	是	是	是	否
YAFFS	是	是	是	否	否

4.2.2 各文件系统 mount 时间的对比

文件系统的内容为8M,采用不同的文件系统mount的时间对比，如图4-3。其中

可读文件系统中squanshfs是最快的，可写文件系统中yaffs2是最快的[34]。

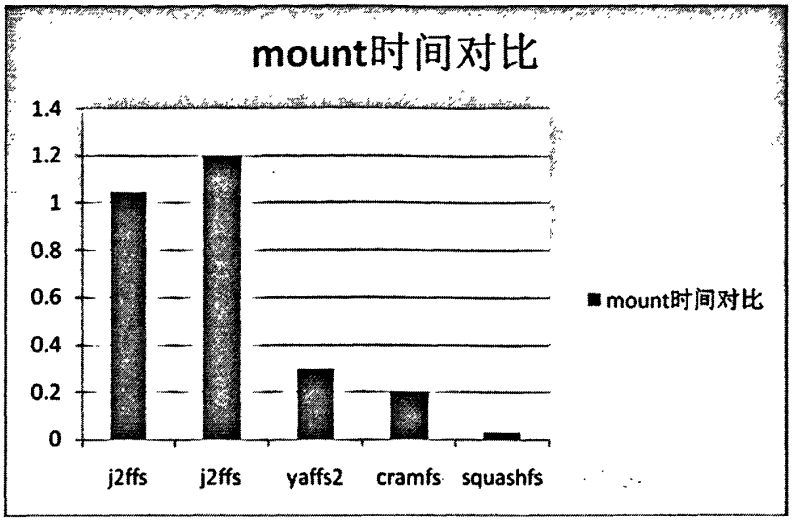


图4-3 流行文件系统Mount时间对比

Figure 4-3 Mount time comparison between popular file systems

4.2.3 选择文件系统

4.2.3.1 选择文件系统的原则

如果系统只配有少量的flash，但是相对而言具有大量的RAM,那么RAM disk或许是最佳选择，因为RAM disk上的文件系统在存储设备上时经过压缩的，因为压缩了元数据，压缩比通常比原生的压缩文件系统高的多。但是RAM disk在运行时必须把文件系统全部解压到内存中运行，需要消耗大量的内存。RAM disk高压缩比的优势却被较高的RAM使用量抵消。此外，RAM disk不能永久性存储[21]。

如果flash容量稍大，可以使用cramfs和squashfs，能提供较高的压缩比，节约存储空间。但是这两种文件系统是可读的，如果用户需要永久保存数据，那么就不太合适了。从压缩比和性能方面综合考虑，在可压缩不可写的文件系统中，squashfs已经可以完全取代cramfs[35]。

YAFFS2是效果很理想的NAND Flash上的文件系统，是专门为NAND Flash设计的嵌入式文件系统，适用于大容量的存储设备。YAFFS文件系统分为文件系统管理层接口、YAFFS内部实现层和NAND接口层，这简化了与系统的接口设计，便于集成到系统中去。它为日志文件系统提供了损耗平衡和掉电保护，保证数据在系统对文件系统修改的过程中发生意外而不被破坏。但YAFFS文件系统不支持

数据压缩，不支持大容量的NAND Flash，为此改进为YAFFS2文件系统。YAFFS2包括了YAFFS 的代码，它利用YAFFS实现对小页Flash的支持，用YAFFS2实现对大页的支持。同时， YAFFS2在内存空间占用 、垃圾回收速度、读/写速度等方面均有大幅提升[36]。

YAFFS/YAFFS2采用一种多策略混合的垃圾回收算法，将贪心策略和随机选择策略按一定比例混合使用：当满足特定的小概率条件时，垃圾回收器会试图随机选择一个可回收的页面；否则，使用贪心策略回收最"脏"的块。这种多策略混合法能够有效地改善贪心策略造成的不均；通过不同的混合比例，控制损耗平均和系统开销之间的平衡。由于NAND的擦除很快，YAFFS将垃圾收集的检查放在写入新页面时进行，而不是采用JFFS那样的后台线程方式，从而简化了设计[37]。

4.2.3.2 确定嵌入式导航系统的文件系统

选择文件系统要综合考虑嵌入式导航系统需求，嵌入式导航设备要求启动速度快，要在几秒钟之内启动，与监控端通信的过程中，要求记录监控端的指令，要想在市场中获得很好的市场份额，成本必须压缩到最小。要达到以上要求，需要可写的启动速度快同时又要求支持压缩的文件系统。

通过以上分析可以知道，单一的文件系统无法满足以上的需求，单一的文件系统无法解决问题时，本文提出了选择多个文件系统的组合，来达到存储空间少，可写，可永久保存、运行性能好的要求。经过比较和测试，本文选择squanshfs、yaffs2、tmpfs三种类型的文件系统组合来达到系统的要求。Squanshfs文件系统具有压缩比高的特点，占用空间小，系统启动速度快，把不会发生改变的文件制作成squanshfs文件映像，用它来做根文件系统。Yaffs2文件系统具有可写的特性，我们把需要修改的且需要永久保存的文件都存储这个文件系统中。tmpfs的特点是可写，但是不会永久保存，系统断电之后文件将不再保存，我们利用这个特性来存储不需要永久保存的要求可写临时文件。所以文件系统的内容需要分成3类，一类是系统的库文件，可执行文件，这类文件是不会发生更改的，我们可以用squashfs文件系统，压缩比高，节约存储空间。一类是会发生更改或者需要被创建，但是不需要保存的临时文件，我们采用tmpfs文件系统，速度快，性能好，系统断电之后文件就不存在，节约了存储空间，同时提高了系统性能。最后一类是会发生更改，但同时需要永久保存的配置文件等，这类用yaffs2文件系统。

4.2.4 文件系统的结构

Linux系统大致有以下一些目录

表4-2 文件系统结构

Table 4-2 File System Structure

目录	内容
bin	必要的用户命令（二进制文件）
boot	引导加载程序 dd的静态文件
dev	设备文件和其他特殊文件
etc	系统配置文件，包括启动文件
home	用户主目录，包括供服务账号所使用的主目录，例如 FTP
lib	必要的连接库，例如C链接库、内核模块
mnt	安装点，用于暂时安装文件系统
opt	附加的软件套件
proc	用来提供内核与进程信息的虚拟文件系统
root	Root用户的主目录
sbin	必要的系统管理员命令
tmp	暂时性的文件
usr	在第二层包含对大多数用户都有用的大量应用程序和文件
var	监控程序和工具程序所存放的可变数据

根据系统要求，不需要boot、root目录，即使留下，也是空的，所以删除。

4.2.5 文件系统的制作

4.2.5.1 文件系统的内容

文件系统的内容大致分为三类，一类是可运行的二进制文件和配置文件，一类是库文件，一类是设备文件。

利用busybox工具来制作文件系统。

busybox被非常形象地称为嵌入式Linux系统中的“瑞士军刀”,因为它将许多常用的UNIX命令和工具结合到了一个单独的可执行程序中。busybox在设计上就充分考虑了硬件资源受限的特殊工作环境。它采用一种很巧妙的办法减少自己的体

积：所有的命令都通过“插件”的方式集中到一个可执行文件中,在实际应用过程中通过不同的符号链接来确定到底要执行哪个操作。例如最终生成的可执行文件为busybox,当为它建立一个符号链接ls的时候,就可以通过执行这个新命令实现列目录的功能。采用单一执行文件的方式最大限度地共享了程序代码,甚至连文件头、内存中的程序控制块等其他操作系统资源都共享了,对于资源比较紧张的系统来说,真是最合适不过了[38]。

在busybox的编译过程中,可以非常方便地加减它的“插件”,最后的符号链接也可以由编译系统自动生成。本文针对系统要求，用busybox建立一个全新的Linux文件系统，具体步骤如下：

1、下载busybox源代码包 http://busybox.net/；并解压

2、进入busybox的文件夹，使用make menuconfig进入设置

3、在build options中有一些选项要注意

Build BusyBox as a static binary (no shared libs):如果选择了这个选项，那么BusyBox将被静态编译，也就是不需要动态链接库的支持就能运行，但是这样会使编译后文件的体积增大。

Do you want to build BusyBox with a Cross Compiler：因为busybox将要运行机器与编译主机的体系结构不同，s3c2440是ARM，与我们的X86主机芯片系统结构不一样，那么这个一定要选上，并且在 Cross Compiler prefix中写上交叉编译器的名字，ARM用arm-Linux-。

4、在Installation Options中 BusyBox installation prefix就是在编译完后运行make install 时，busybox将被安装的地址。

5、其它的选项按照需要选择，不要把不需要的选项加入到编译选项中来。

6、完成后保存，然后运行make。

7、编译没有问题后运行make install这样编译好的busybox就会自动安装到busybox installation prefix中，为了降低磁盘占用空间，一般是三个文件夹bin sbin usr和一个文件Linuxrc。这样busybox的编译就完成了。

8、准备一个空文件夹将busybox installation prefix中的东西拷过来，另外再建几个Linux需要的文件夹。

mkdir lib dev etc　home proc　tmp var mnt

9、把交叉编译工具的库文件拷贝到lib文件夹中。

10、在dev文件夹下创建所需要的设备文件节点。

11、在/etc目录下创建启动脚本。

经过以上11个步骤，我们所需要的文件系统的内容已经准备完毕。

为了降低存储空间，同时又不降低性能，我们采用了多种文件系统组合的方式来处理，根据是否可写和是否永久保存，我们应该把文件做不同的处理，我们把var目录和etc目录下的部分文件做成yaffs2文件系统，tmp作为tmpfs文件系统，其他做成squashfs文件系统，并作为根文件系统使用。我们把不同的文件分开，那又应该如何访问呢？实现很简单，我们把yaffs2文件系统挂载到squashfs根文件系统的/mnt目录中，在squashfs文件系统建立软链接，如：根文件系统中的var链接到/mnt/var命令如下：ln /mnt/var /var –s。访问/var中的目录中的内容将会自动到/mnt/var目录中寻找。

4.2.5.2 制作 squashfs 文件系统

1、下载squanshfs工具包，工具包内包含内核补丁和制作工具源代码

2、给系统打上补丁，

3、进入make menuconfig配置界面，把squashfs文件系统加入到编译选项中。

4、make 内核，此时的内核已经支持了squashfs文件系统了。

5、制作mksquashfs工具，

6、使用mksquashfs制作img文件，mksquashfs /rootfilesystem rootfilesystem.squashfs

7、通过boot loader的烧写功能，把rootfilesystem.squashfs烧入/dev/mtd2

Squashfs类型的根文件系统制作完毕。

4.2.5.3 制作 yaffs2 文件系统

制作yaffs2文件系统需要同样的步骤，打补丁，编译img文件，烧写的过程[43]。跟制作squanshfs类似，这里不具体叙述。

不同的地方在于:

1、烧写的位置在/dev/mtd3

2、yaffs2文件系统不是根文件系统，所以需要挂载到根文件系统的一个挂载点上。我们挂载到/mnt上，挂载命令：

mount /mnt /dev/mtdblock3 -t yaffs2

由上就建立了一个可写文件系统，可以投入使用。

4.2.5.4 挂载 tmpfs 文件系统

1、首先编译内核增加“虚拟内存文件系统支持(Virtual memory tmpfs)”。

2、Mount命令创建tmpfs文件系统

mount tmpfs /tmp -t tmpfs

因为tmpfs不同于大部分的文件系统，如ext3，ext2，XFS，JFS等等，tmpfs并不是存在于一个底层的块设备上面。tmpfs直接建立在虚拟内存子系统上，所以只需要mount命令就可以创建tmpfs文件系统了，执行这个命令后，一个新的tmpfs就安装在/tmp，文件系统立即被安装并且可以使用，类型是tmpfs。这和Linux虚拟磁盘的使用大相径庭。标准的Linux虚拟磁盘是块设备，所以在使用它之前必须选择文件系统将其格式化。相反，tmpfs本身是一个文件系统，所以只要mount就能使用了。

4.2.6 文件系统运行测试结果

本系统已成功建立和实现了一个适用于嵌入式导航设备的文件系统，将 squashfs 文件系统映像和 yaffs2 映像烧写入/dev/mtd2，/dev/mtd3。启动系统，启动信息如图 4-4，这里将不在出现 VFS: Cannot open root device“mtdblock2” or unknown-block(0,0)错误信息。进入控制台，系统运行正常。

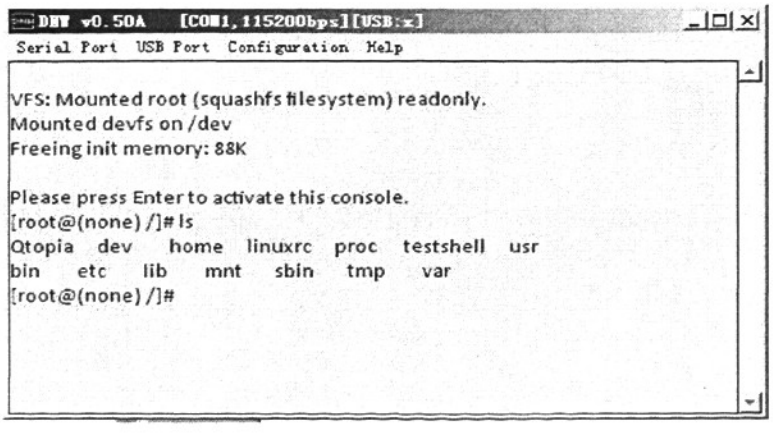


图4-4 文件系统测试结果图

Figure 4-4 File System test results

4.3 本章小结

本章首先介绍了Linux作为嵌入式操作系统的优势及其组成，研究了操作系统移植的一般原理和移植过程，实现了支持嵌入式导航系统的Linux内核的移植，并给出关键代码，通过测试验证了内核移植的正确性，然后介绍了文件系统的选择原则，根据嵌入式导航系统的实际需求，提出了文件系统优化建立方法，给出了3种文件系统的制作和移植的关键步骤，最后对建立的文件系统进行了测试。

第五章 导航系统的驱动程序设计

5.1 Linux 设备驱动简介

设备驱动程序在Linux内核中扮演着特殊的角色。它们是一个个独立的"黑盒子"，使某个特定的硬件响应一个定义良好的内部编程接口，这些接口完全隐藏了设备的工作细节。用户的操作通过一组标准化得调用执行，而这些调用独立于特定的驱动程序。将这些调用映射到作用于实际硬件的设备特有操作上，则是设备驱动程序的任务。

在编写驱动程序时，必须注意不要给用户强加任何特定策略。因为不同的用户有不同的需求，驱动程序应该处理如何使硬件可用的问题，而将怎样使用硬件的问题留给上层应用程序。

总之，设备驱动程序屏蔽了具体硬件的操作过程，使用户可以直接调用具体硬件的接口而不用去关心具体硬件的操作过程，有效地提高了编程效率。设备驱动的一个基本特征是设备处理的抽象概念。所有硬件设备都被看成普通文件，可以通过操纵普通文件相同的标准系统调用来打开、关闭、读取和写入设备[39]。

5.1.1 设备分类

Linux系统将设备分成三种基本类型，字符模块、块模块和网络模块。这三种类型如下：

(1) 字符设备

字符（char）设备是个能够像字节流一样被访问的设备，由字符设备驱动程序来实现这种特性。字符设备驱动程序通常至少实现open， close，read和write系统调用。字符设备可以通过文件系统节点来访问，比如/dev/tty1和/dev/lp0等。这些设备文件和普通文件之间的唯一差别在于对普通文件的访问可以前后移动访问位置，而大多数字符设备是一个只能顺序访问的数据通道。

(2) 块设备

和字符设备类似，块设备也是通过/dev目录下的文件系统节点来访问。块设备

上能够容纳文件系统。在大多数Unix系统上，进行I/O操作时块设备每次只能传输一个或多个完整的块，而每块包含512字节（或2的更高次幂字节的数据）。Linux可以让应用程序像字符设备一样地读写块设备，允许一次传递任意多字节的数据。因而，块设备和字符设备的区别仅仅在于内核内部管理数据的方式，也就是内核及驱动程序之间的软件接口，而这些不同对用户来讲是透明的。在内核中，和字符驱动程序相比，块驱动程序具有完全不同的接口。

(3) 网络设备

任何网络事物都经过一个网络接口形成，即一个能够和其他主机交换数据的设备。通常，接口是个硬件设备，但也可能是个纯软件设备，比如回环（loopback）接口，网络接口由内核中的网络子系统驱动，负责发送和接收数据包，但它不需要了解每项事务如何映射到实际传送的数据包。许多网络连接（尤其是使用TCP协议的连接）是面向流的，但网络设备却围绕数据包的传输和接收而设计，网络驱动程序不需要知道各个连接的相关信息，只需要处理数据包即可。

由于不是面向流的设备，因此将网络接口映射到文件系统中的节点(比如/dev/ttyS1)比较困难。Linux访问网络接口的方法仍然是给它们分配一个唯一的名字（比如eth0），但是这个名字在文件系统中不存在对应的节点。内核和网络设备驱动程序间的通信，完全不同于内核和字符以及块驱动程序之间的通信，内核调用一套和数据包传输相关的函数而不是read, write等[40]。

5.1.2 可装载模块

Linux有一个很好的特性：内核提供的特性可在运行时进行扩展。这意味着当系统启动并运行时，我们可以向内核添加功能（当然也可以移除功能）

可在运行时添加到内核中的代码被称为“模块”。Linux内核支持好几种模块类型，包括但不限于设备驱动程序。每个模块由目标代码组成，我们可以使用insmod程序将模块连接到正在运行的内核，也可以用rmmod程序移除连接。

不管内核来自哪里，要想为2.6.x内核构造模块，还必须在自己的系统中配置并构造好内核树[12]。

5.1.3 编译和装载

模块建立之后，下一步是加载到内核。如前所述，insmod 可以完成这个工作。

这个程序加载模块的代码段和数据段到内核，接着，执行一个类似 ld 的函数，它连接模块中任何未解析的符号连接到内核的符号表上。与连接器不同，内核不修改模块的磁盘文件，而是内存内的拷贝。insmod 接收许多命令行选项，它能够安排值给你模块中的参数，在连接到当前内核之前。因此，如果一个模块正确设计了，它能够在加载时配置； 加载时配置比编译时配置给了用户更多的灵活性，有时仍然在用。

modprobe工具值得快速提及一下。modprobe，如同insmod，加载一个模块到内核。它的不同在于它会查看要加载的模块，看是否它引用了当前内核没有定义的符号。如果发现有，modprobe在定义相关符号的当前模块搜索路径中寻找其他模块。当modprobe找到这些模块（要加载模块需要的），它也把它们加载到内核。如果你在这种情况下代替以使用insmod ，命令会失败，在系统日志文件中留下一条"unresolved symbols"消息。

模块可以用rmmod工具从内核去除。注意，如果内核认为模块还在用（就是说，一个程序仍然有一个打开文件对应模块输出的设备），或者内核被配置成不允许模块去除，模块去除会失败。可以配置内核允许"强行"去除模块，甚至在它们看来是忙的，也是可以去除模块的。如果我们在某种情况下希望利用这种特性，则重新引导系统可能是更加合适的做法。

lsmod程序生成一个内核中当前加载的模块的列表，还提供一些其他信息，例如一个特定模块使用了其他模块。lsmod通过读取 /proc/modules虚拟文件来获取这些信息的[13]。

5.1.4 嵌入式导航系统的设备驱动

嵌入式导航系统需要实现导航，通信等功能，需要底层设备的支持，设备的驱动程序非常重要。根据系统的需求，必须设计与实现以下几种设备的驱动程序。

1、串口驱动，硬件平台提供 3 路串口，GPS、GPRS 使用其中 2 路，另外一个用于串口控制终端。为了实现导航信息的接收功能与通信功能，必须实现串口驱动。

2、触摸屏驱动，硬件平台没有使用键盘，人机交互的工作就全部落在触摸屏上，完善稳定的触摸屏驱动非常重要。

3、USB 驱动，系统实现外扩数据必须使用 USB host 接口。USB 驱动是必不可少的。

4、Nandflash 驱动，嵌入式导航系统的 bootloader，内核，导航软件全部存放在 nandflash 上，文件系统也是基于 nandflash 来实现的，BSP 软件系统必须要有稳定的 nandflash 驱动。

本系统需要设计与实现串口驱动、USB 驱动、nandflash 驱动和触摸屏驱动，在第四章，操作系统移植的过程中，已经加入了 nandflash 的驱动代码，成功实现了 nandflash 驱动。其他三种设备驱动在以下章节详细介绍。

5.2 USB 驱动程序开发

5.2.1 USB 总线优点

（1）使用简单

所用 USB 系统的接口一致，连线简单。系统可对设备进行自动检测和配置，支持热插拔。新添加设备系统不需要重新启动。

（2）应用范围广

USB 系统数据报文附加信息少，带宽利用率高，可同时支持同步传输和异步传输两种传输方式。一个 USB 系统最多可支持 127 个物理设备。USB 设备的带宽可从几 Kbps 到几 Mbps （在 USB2.0 版本，最高可达几百 Mbps）。一个 USB 系统可同时支持不同速率的设备。

（3）较强的纠错能力

USB 系统可实时地管理设备插拔。在 USB 协议中包含了传输错误管理、错误恢复等功能，同时根据不同的传输类型来处理传输错误。

（4）总线供电

USB 总线可为连接在其上的设备提供 5V 电压/100mA 电流的供电，最大可提供 500mA 的电流。USB 设备也可采用自供电方式。

（5）低成本

USB 接口电路简单，易于实现，特别是低速设备。USB 系统接口/电缆也比较简单，成本比串口/并口低[41]。

5.2.2 USB 的传输方式

在 USB 的数据传送的方式下，有四种的传输方式：控制(Control)、同步(isochronous)、中断(interrupt)、大量(bulk)。如果你是从硬件开始来设计整个的系统，你还要正确选择传送的方式，而作为一个驱动程序的书写者，就只需要弄清楚他是采用的什么工作方式就行了。通常所有的传送方式下的主动权都在 PC 边，也就是 host 边。

控制(Control)方式传送：控制传送是双向传送，数据量通常较小。USB 系统软件用来主要进行查询、配置和给 USB 设备发送通用的命令。控制传送方式可以包括 8、16、32 和 64 字节的数据，这依赖于设备和传输速度。控制传输典型地用在主计算机和 USB 外设之间的端点(Endpoint)0 之间的传输，但是指定供应商的控制传输可能用到其它的端点。

同步(isochronous)方式传送：同步传输提供了确定的带宽和间隔时间(latency)。它被用于时间严格并具有较强容错性的流数据传输，或者用于要求恒定的数据传送率的即时应用中。例如执行即时通话的网络电话应用时，使用同步传输模式是很好的选择。同步数据要求确定的带宽值和确定的最大传送次数。对于同步传送来说，即时的数据传递比完美的精度和数据的完整性更重要一些。

中断(interrupt)方式传送：中断方式传输主要用于定时查询设备是否有中断数据要传送。设备的端点模式器的结构决定了它的查询频率，从 1 到 255ms 之间。这种传输方式典型的应用在少量的分散的、不可预测数据的传输。键盘、操纵杆和鼠标就属于这一类型。中断方式传送是单向的并且对于 host 来说只有输入的方式。

大量(bulk)传送：主要应用在数据大量传送传送和接受数据上，同时又没有带宽和间隔时间要求的情况下，要求保证传输。打印机和扫描仪属于这种类型。这种类型的设备适合于传输非常慢和大量被延迟的传输，可以等到所有其它类型的数据的传送完成之后再传送和接收数据。

USB 将其有效的带宽分成各个不同的桢(frame)，每桢通常是 1ms 时间长。每个设备每桢只能传送一个同步的传送包。在完成了系统的配置信息和连接之后，USB 的 host 就会对不同的传送点和传送方式做一个统筹安排，用来适应整个的

USB 的带宽。通常情况下，同步方式和中断方式的传送会占据整个带宽的 90%，剩下的就安排给控制方式传送数据。

5.2.3 Linux USB 驱动层次

USB 采用树形拓扑结构，主机侧和设备侧的 USB 控制器分别称为主机控制器（Host Controller）和 USB 设备控制器（UDC），每条总线上只有一个主机控制器，负责协调主机和设备间的通信，而设备不能主动向主机发送任何消息。如图 5-1 所示，在 Linux 系统中，USB 驱动可以从两个角度去观察，一个角度是主机侧，一个角度是设备侧。

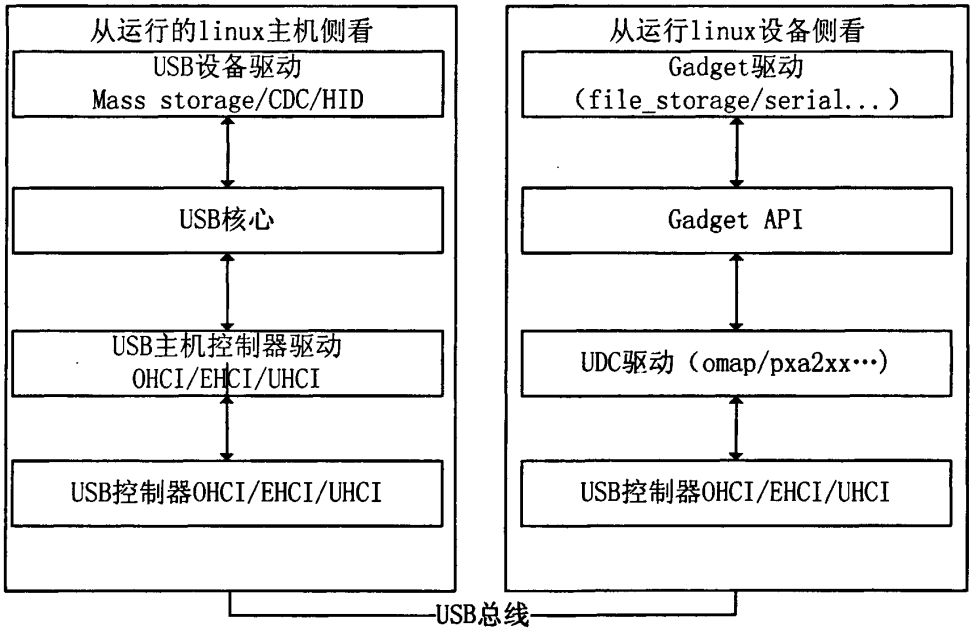


图 5-1 Linux USB 驱动总体结构

Figure 5-1 the overall structure of Linux USB drivers

如图 5-1 的左侧所示，从主机侧的观念去看，在 Linux 驱动中，USB 驱动处于最底层的是 USB 主机控制器硬件，在其之上运行的是 USB 主机控制器驱动，主机控制器之上为 USB 核心层，再上层为 USB 设备驱动层（插入主机上的 U 盘、鼠标、USB 转串口等设备驱动）。因此，在主机侧的层次结构中，要实现的 USB 驱动包括两类：USB 主机控制器驱动和 USB 设备驱动，前者控制插入其中的 USB 设备，后者控制 USB 设备如何与主机通信。Linux 内核 USB 核心负责 USB 驱动管理和协议处理的主要工作。主机控制器驱动和设备驱动之间的 USB 核心

非常重要，其功能包括：通过定义一些数据结构、宏和功能函数，向上为设备驱动提供编程接口，向下为 USB 主机控制器驱动提供编程接口；通过全局变量维护整个系统的 USB 设备信息；完成设备热插拔控制、总线数据传输控制等。

如图 5-1 的右侧所示，Linux 内核中 USB 设备侧驱动程序分为 3 个层次：UDC 驱动程序、Gadget API 和 Gadget 驱动程序。UDC 驱动程序直接访问硬件，控制 USB 设备和主机间的底层通信，向上层提供与硬件相关操作的回调函数。当前 Gadget API 是 UDC 驱动程序回调函数的简单包装。Gadget 驱动程序具体控制 USB 设备功能的实现，使设备表现出"网络连接"、"打印机"或"USB Mass Storage"等特性，它使用 Gadget API 控制 UDC 实现上述功能。Gadget API 把下层的 UDC 驱动程序和上层的 Gadget 驱动程序隔离开，使得在 Linux 系统中编写 USB 设备侧驱动程序时能够把功能的实现和底层通信分离[42]。

下面将重点讲解从主机侧角度看到的 USB 主机控制器驱动与 USB 设备驱动，关于设备侧的 Linux 驱动，由于 USB 设备很多，这里将不详细介绍。

5.2.4 S3C2440 USB 控制器方框图

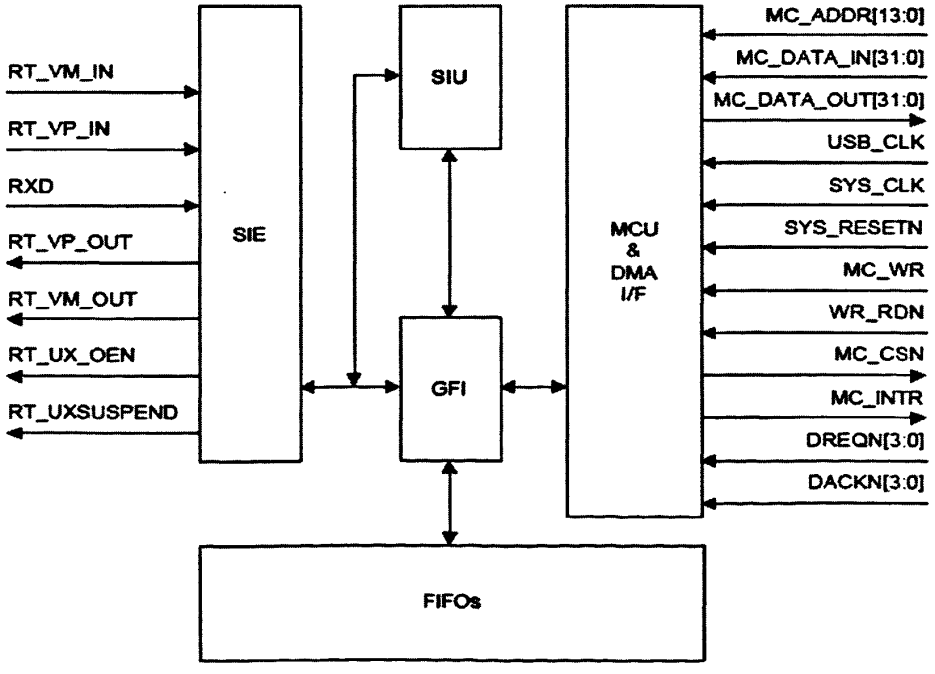


图 5-2 S3C2440 USB 控制器方框图

Figure 5-2 S3C2440 USB controller block diagram

5.2.5 S3C2440 USB 主机驱动

S3C2440 内部集成了一个 USB 主机控制器，完全兼容 OCHI 1.0、USB 1.1 标准，支持低速和全速 USB 设备，从基地址 0x49000000 开始分别提供了 OHCI 的 HcRevision、HcControl、HcCommonStatus、HcInterruptStatus、HcInterruptEnable、HcInterruptDisable、HcHCCA、HcPeriodCuttentED、HcControlHeadED、HcControlCurrentED、HcBulkHeadED、HcBulk CurrentED、HcDoneHead、HcRmInterval、HcFmRemaining、HcFmNumber、HcPeriodicStart、HcLSThreshold、HcRhDescriptorA、HcRhDescriptorB、HcRhStatus、HcRhPortStatus1、HcRhPortStatus2 寄存器。

S3C2440 主机控制器驱动 hc_driver 结构体中的大多数成员函数都是通用的 ohci_xxx()函数，而 start()、hub_status_data()、hub_control()函数则针对 s3c2440 而编写的。

s3c2440 主机控制器驱动的 hc_driver 结构体

```
static const struct hc_driver ohci_s3c2440_hc_driver =
{
 .description = hcd_name,
 .product_desc = "S3C24XX OHCI",
 .hcd_priv_size = sizeof(struct ohci_hcd),
 /* 通用硬件联接 */
 .irq = ohci_irq,
 .flags = HCD_USB11 | HCD_MEMORY, //USB 1.1 标准，hc 寄存器位于内存

 /* 基本的生命周期操作 */
 .start = ohci_s3c2440_start,
 .stop = ohci_stop,
 /* 管理 I/O 请求和相关的设备资源 */
 .urb_enqueue = ohci_urb_enqueue,
 .urb_dequeue = ohci_urb_dequeue,
 .endpoint_disable = ohci_endpoint_disable,
 /* 调度支持 */
 .get_frame_number = ohci_get_frame,
 /* 根 Hub 支持 */
 .hub_status_data = ohci_s3c2440_hub_status_data,
 .hub_control = ohci_s3c2440_hub_control,
 #ifdef CONFIG_PM
 .bus_suspend = ohci_bus_suspend,
 .bus_resume = ohci_bus_resume,
 #endif
 .start_port_reset = ohci_start_port_reset,
};
```

hc_driver 的 start()成员函数用于初始化 OHCI 并启动主机控制器。

s3c2440 主机控制器驱动的 start()函数

```
static int ohci_s3c2440_start (struct usb_hcd *hcd)
{
 struct ohci_hcd *ohci = hcd_to_ohci (hcd);
 int ret;
 if ((ret = ohci_init(ohci)) < 0) //初始化 ohci_hcd
 return ret;
 if ((ret = ohci_run (ohci)) < 0) { //启动 ohci_hcd
 err ("can't start %s", hcd->self.bus_name);
 ohci_stop (hcd);
 return ret;
}
 return 0;
 }
```

ohci_s3c2440_hub_status_data()函数

```
static int
ohci_s3c2440_hub_status_data (struct usb_hcd *hcd, char *buf)
{
struct s3c2440_hcd_info *info = to_s3c2440_info(hcd);
struct s3c2440_hcd_port *port;
int orig;
int portno;
orig = ohci_hub_status_data (hcd, buf);
if (info == NULL)
return orig;
port = &info->port[0];
/* mark any changed port as changed */
for (portno = 0; portno < 2; port++, portno++) {
if (port->oc_changed == 1 &&
port->flags & S3C_HCDFLG_USED) {
dev_dbg(hcd->self.controller,
"oc change on port %d\n", portno);
if (orig < 1)
orig = 1;
buf[0] |= 1<<(portno+1);
}
}
return orig;
}
```

hc_driver 的 hub_control()成员函数 ohci_s3c2440_hub_control()中的主体是调用通用的 ohci_hub_control()函数。函数不一一举例。

5.3 触摸屏驱动程序开发

5.3.1 触摸屏硬件实现方案

SPI 接口是 Motorola 推出的一种同步串行接口，采用全双工、四线通信系统，S3C2440 是三星推出的自带触摸屏接口的 ARM920T 内核芯片，ADS7843 为 Burr-Brown 生产的一款性能优异的触摸屏控制器。本文采用 SPI 接口的触摸屏控制器 ADS7843 外接四线电阻式触摸屏，这种方式最显著的特点是响应速度更快、灵敏度更高，微处理器与触摸屏控制器间的通讯时间大大减少，提高了微处理器的效率。ADS7843 与 S3C2440 的硬件连接如图 5-3 所示，鉴于 ADS7843 差分工作模式的优点，在硬件电路中将其配置为差分模式。

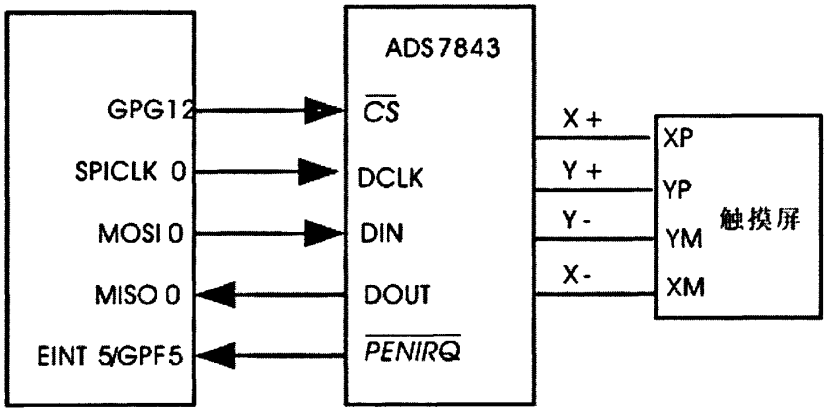


图 5-3 触摸屏硬件连接图

Figure 5-3 Touch-screen hardware connection diagram

5.3.2 触摸屏驱动程序

这里采用可安装模块方式开发调试触摸屏驱动程序。

触摸屏驱动程序中重要数据结构

```
typedef struct {
  unsigned short pressure;
  unsigned short x;
  unsigned short y;
  unsigned short pad;
} TS_RET;
typedef struct {
```

```
 unsigned int PenStatus;
 TS_RET buf[MAX_TS_BUF];
 unsigned int head, tail;
 wait_queue_head_t wq;
 spinlock_t lock;
} TS_DEV;
static struct file_operations s3c2440_fops = {
  owner: THIS_MODULE,
  open: s3c2440_ts_open,
  read: s3c2440_ts_read,
 release: s3c2440_ts_release,
poll: s3c2440_ts_poll,    };
```

在程序中有三个重要的数据结构：用于表示笔触点数据信息的结构 TS_RET，表示 ADS7843 中有关触摸屏控制器信息的结构 TS_DEV，以及驱动程序与应用程序的接口 file_operations 结构的 s3c2440_fops。

TS_RET 结构体中的信息就是驱动程序提供给上层应用程序使用的信息，用来存储触摸屏的返回值。上层应用程序通过读接口，从底层驱动中读取信息，并根据得到的值进行其他方面的操作。

TS_DEV 结构用于记录触摸屏运行的各种状态，PenStatus 包括 PEN_UP、PEN_DOWN 和 PEN_FLEETING。buf[MAX_TS_BUF]是用来存放数据信息的事件队列，head、tail 分别指向事件队列的头和尾。程序中的笔事件队列是一个环形结构，当有事件加入时，队列头加一，当有事件被取走时，队列尾加一，当头尾位置指针一致时读取笔事件的信息，进程会被安排进入睡眠。wq 等待队列，包含一个锁变量和一个正在睡眠进程链表。当有好几个进程都在等待某件事时，Linux 会把这些进程记录到这个等待队列。它的作用是当没有笔触事件发生时，阻塞上层的读操作，直到有笔触事件发生。lock 使用自旋锁，自旋锁是基于共享变量来工作的，函数可以通过给某个变量设置一个特殊值来获得锁。而其他需要锁的函数则会循环查询锁是否可用。MAX_TS_BUF 的值为 16，即在没有被读取之前，系统缓冲区中最多可以存放 16 个笔触数据信息。

s3c2440_fops 就是内核对驱动的调用接口，完成了将驱动函数映射为标准接口。上面的这种特殊表示方法不是标准 C 的语法，而是 GNU 编译器的一种特殊扩展，它使用名字进行结构字段的初始化，它的好处体现在结构清晰，易于理解，并且避免了结构发生变化带来的许多问题。

init_module 函数

这是模块的入口函数。在函数内部通过 s3c2440_ts_init( )实现模块的初始化工作。在本设计中设备与系统之间以中断方式进行数据交换。整个触摸屏的驱动程序处理比较复杂，而且耗时较长，因而触摸屏驱动程序不可能在中断服务程序中完成。在 Linux 操作系统中一般把中断处理切为两个部分或两半。中断处理程序是上半部——接收到一个中断，它就立即开始执行，但只做有严格时限的工作，例如对接收的中断进行应答或复位硬件。这些工作都是在所有中断被禁止的情况下完成的，能够被允许稍后完成的工作会推迟到下半部去。在 Linux 中下半部的实现有多种机制。按触摸屏时，从 ADS7843 输出的数值有一个抖动过程，即从 ADS7846 输出的数值有一个不稳定时期，这个过程大约为 10ms。所以中断处理程序的下半部处理函数采用内核定时器机制，使下半部在中断发生 50ms 后再作处理。这样有效地避开了 ADS7843 输出值的不稳定时期，使中断服务程序和中断处理任务串行化，达到了处理时间较长的触摸屏事件的目的。驱动程序通过 request_irq 函数注册并激活一个中断处理程序，以便处理中断。

int reguest_irq(unsigned int irq，void(*handler)(int，void *，struct pt_regs *)，unsigned long irq_flags，const char *dev_name，void *dev_id)

参数 irq 表示所要申请的中断号；handler 为向系统登记的中断处理子程序，中断产生时由系统来调用；dev_name 为设备名；dev_id 为申请时告诉系统的设备标识符；irq_flags 是申请时的选项，它决定中断处理程序的一些特性，其中最重要的是中断处理程序是快速处理程序还是慢速处理程序。

触摸屏控制器 ADS7843 的中断输出通过外部中断 5 接在中断控制器上，当触摸屏上有触摸事件发生时，会引发中断号为 IRQ_EINT5 的中断服务程序 s3c2440_isr_tc()。图 5-4 所示为该中断处理程序的流程图。

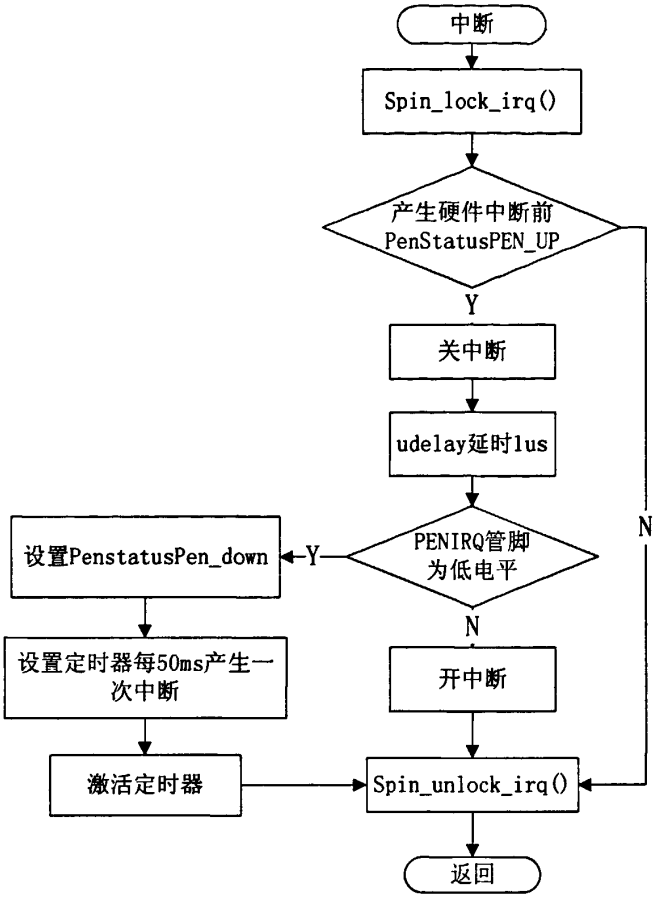


图 5-4 触摸屏硬件中断处理程序流程图

Figure 5-4 Touch-screen hardware interrupt processing flow chart

在 s3c2440_isr_tc()中设定了定时器的定时时间为 50ms，并立即激活。因此有触摸屏硬件中断的情况下 50ms 后就会引发定时中断，中断服务程序为 ts_timer_handler()，这个程序实现了触摸屏中断的下半部，即在过了抖动时间之后如果触摸屏确实有有效事件发生则采集触摸屏坐标，并将定时器的时间重新设为 100ms 并重新激活，这样做的目的是如果触摸笔是拖动的情况，以后每 100ms 采集一次坐标值，并存入缓冲区，如果不是拖动在采集一次坐标值之后，在第二次进入 ts_timer_handler()时，查询管脚的状态值，则变为高电平，就将触摸屏状态 tsdev.PenStatus 设为 PEN_UP，并释放定时器，为下次触摸屏事件做好准备，定时中断服务程序流程图如图 5-5 所示。

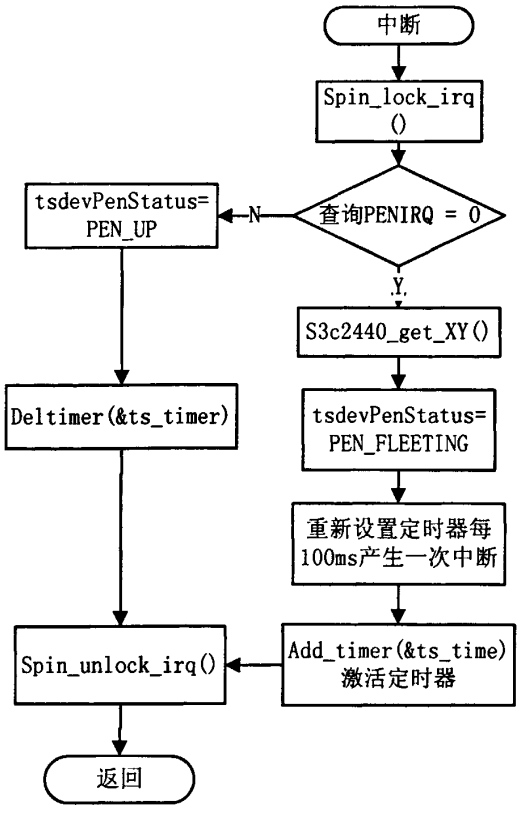


图 5-5 定时中断服务程序流程图

Figure 5-5 Timer interrupt service routine flowchart

在 s3c2440_ts_init()中的另一个重要任务是执行接口函数 s3c2440_ts_open()，在这个函数中初始化缓冲区的头尾指针、触摸屏状态变量及触摸屏事件等待队列。

module_exit()

该函数调用 s3c2440_ts_exit()，主要任务是撤销驱动程序向内核的登记以及释放申请的中断资源。

接口函数 s3c2440_ts_read( )

这个函数实现的任务是将事件队列从设备缓存中读到用户空间的数据缓存中。实现的过程主要是通过一个循环，只有在事件队列的头、尾指针不重合时，才能成功的从 tsdev.tail 指向的队列尾部读取到一组触摸信息数据，并退出循环。否则调用读取函数的进程就要进入睡眠。

坐标读取函数 s3c2440_get_XY

在定时器中断处理程序中，当查询到与相连的 EINT5/GPF5 为低电平时，即表示有有效事件，应该调用 s3c2440_get_XY()函数采集笔触信息。

ADS7843 有多种转换时序，时序规定了芯片与设备及 CPU 间是如何配合工作的。设计中采用 16 个时钟周期启动一次转换的坐标转换方式。坐标的读取是通过多次采集取平均值的方法，以 X 坐标的读取为例，其读取过程如图 5-6 所示。循环过程中的每一步都在 8 个时钟周期内完成，数据的处理严格按照时序进行，Y 坐标的采集与 X 坐标类似。

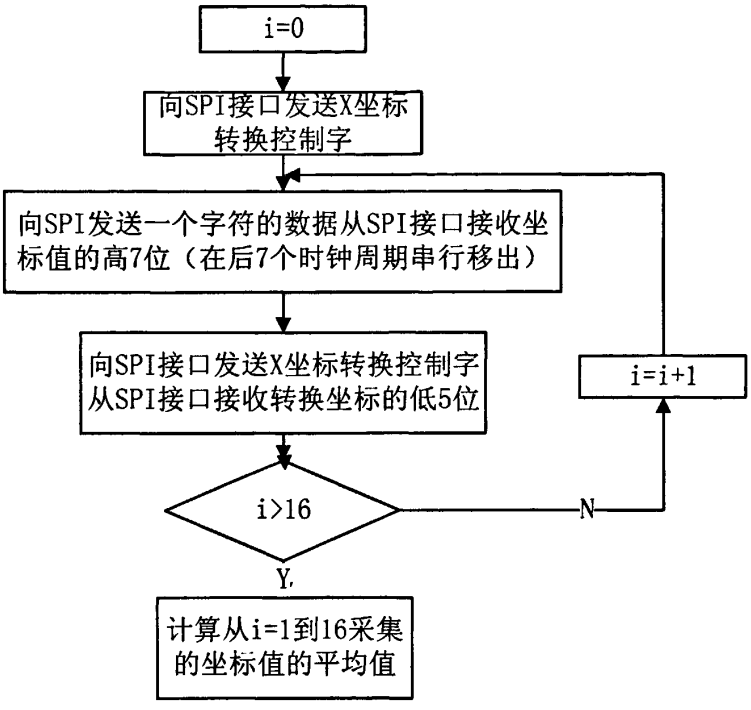


图 5-6 X 坐标采集流程

Figure 5-6 X coordinate acquisition process

在触摸屏的设计中，抗干扰设计是难点和重点，直接关系到触摸屏的工作性能。实验发现坐标采集时，丢弃第一次采集值读取的坐标转换值效果较好。数据稳定性和系统负载的角度看，效果良好。同时通过修改程序内部的定时器时钟频率可以改变笔在屏上移动所产生的数据量。

5.4 串口驱动与 GPRS 拨号

5.4.1 S3C2410 芯片串口控制器方框图

S3C2410 提供了 3 个独立的 UART 控制器，每个控制器都可以工作在 Interrupte 模式或 DMA 模式，每个串口都有一个波特率发生器、接收寄存器、发送寄存器和一个控制单元，另外，还有两个 16 字节的 FIFO 寄存器作为发送和接收的缓冲装置，支持的最高波特率可达到 230.4Kbit/s[45]。其方框图如图 5-7 所示。

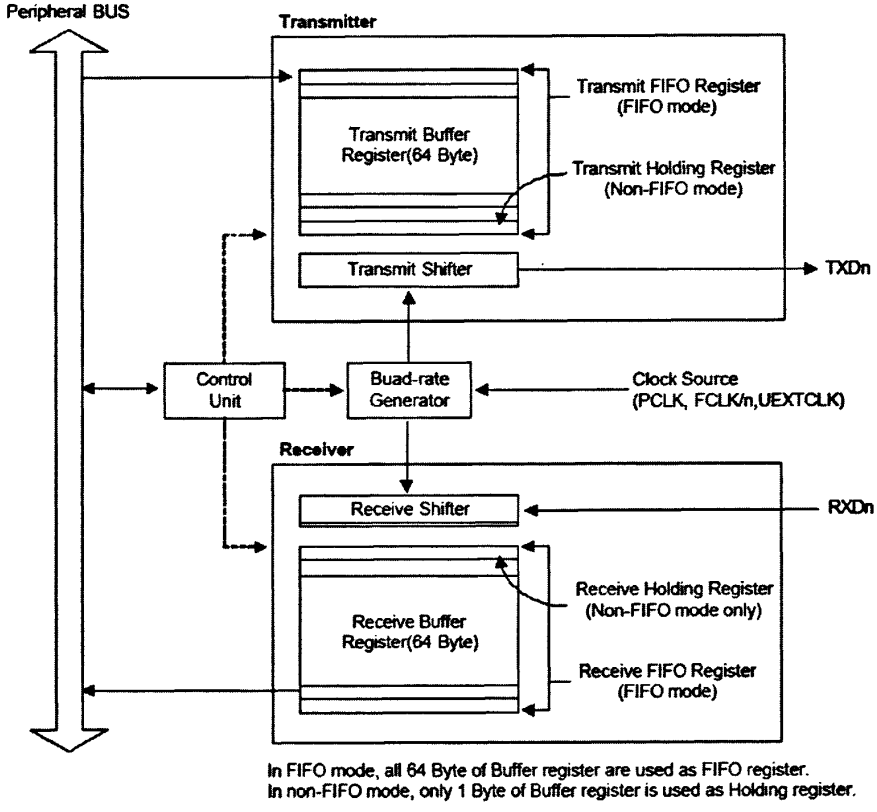


图 5-7 串口方框图

Figure 5-7 UART Block Diagram

在 S3C2440 中，对串口的控制是通过设置相应的控制寄存器来实现的，其常用的寄存器主要有以下几个：

（1）ULCONn 寄存器：主要用来设置串口工作模式，包括数据位长度、停止位个数，数据校验方式以及模式等。S3C2440 支持三种校验方式，分别是奇校验、偶校验、无校验。

（2）UCONn 寄存器：该寄存器涉及到工作模式（中断，DMA，轮询），以及时钟的选择等。

（3）UFCONn 寄存器：主要用于 FIFO 控制，FIFO 使能，FIFO 中断的触发级别和 FIFO 复位。

（4）UFSTATn 寄存器：显示 FIFO 的状态。

（5）UTRSTATn 寄存器：串口的状态寄存器，用于指示串口是否接收或发送完毕。

（6）UTXHn 和 URXHn 寄存器：发送和接收寄存器。

（7）UBRDIVn 寄存器：波特率设置寄存器，用于对时钟分频，产生需要的波特率。其值通过如下计算确定：

UBRDIVn=(int)(PCLK/(bps*16))-1

或者

UBRDIVn=(int)(UCLK/(bps*16))-1

其中 PCLK 或 UCLK 是选择的时钟，bps 是需要的波特率。

比如，波特率是 115200bps，UART 时钟是 40MHZ，UBRDIVn 是

UBRDIVn = (int)(40000000 / (115200 x 16)) -1

= (int)(21.7) -1

= 22 -1 = 21

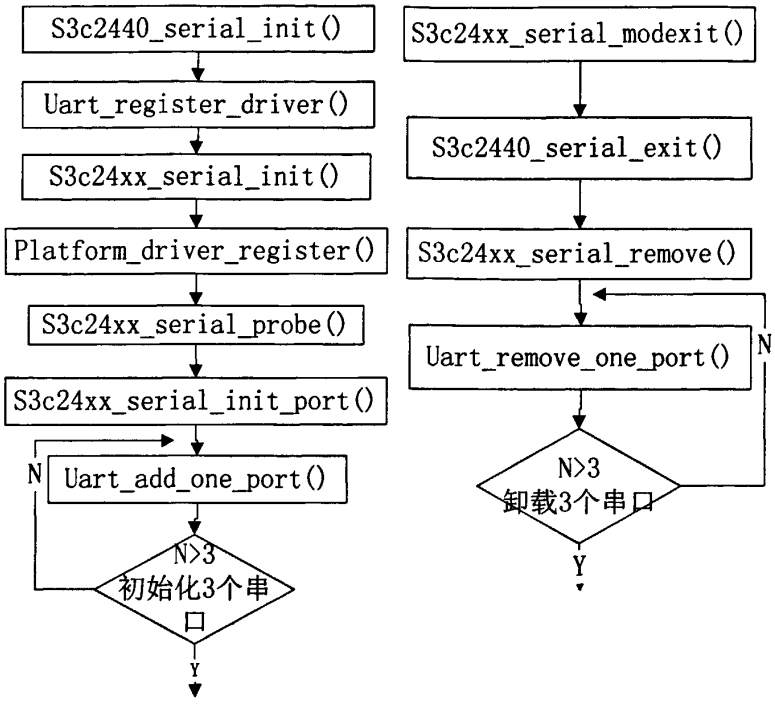
（8）UMCONn 寄存器：主要用于 modem 控制，设置是否使用 RTS 流控等。

5.4.2 串口驱动设计

下面分析串口驱动的设计方法。其具体的编程实现如下：

（1） S3C2440 串口驱动的初始化与释放

S3C2440 串口驱动的模块加载函数 s3c2440_serial_modinit(void)实现流程和串口释放流程如图 5-8 所示：



(a) 串口驱动初始化流程　　(b) 串口驱动释放流程

图 5-8 串口驱动初始化和释放流程

Figure 5-8 UART driver initialization and release processes

（2） S3C2440 串口数据收发

S3C2440 串口驱动 uart_ops 结构体的 startup()成员函数 s3c24xx_serial_startup()用于启动端口，申请端口的发送、接收中断，使能端口的发送和接收。s3c24xx_serial_shutdown()，其释放中断，禁止发送和接收。s3c24xx_serial_tx_empty()用于判断发送缓冲区是否为空，s3c24xx_serial_start_tx()用于启动发送而 s3c24xx_serial_stop_tx()用于停止发送。

S3C2440 串口驱动收发关系最紧密的是串口驱动的接收中断处理函数和发送中断处理函数。接收中断函数原理上与发送中断函数类似，这里给出发送中断函数的关键代码：

```
static irqreturn_t s3c24xx_serial_tx_chars(int irq, void *id, struct pt_regs *regs)
{
    struct s3c24xx_uart_port *ourport = id;
    struct uart_port *port = &ourport->port;
    struct circ_buf *xmit = &port->info->xmit;
    int count = 256;
    if (port->x_char) {
        wr_regb(port, S3C2410_UTXH, port->x_char);
```

```
        port->icount.tx++;
        port->x_char = 0;
        goto out;
    }
    /* if there isnt anything more to transmit, or the uart is now
     * stopped, disable the uart and exit
    */
    if (uart_circ_empty(xmit) || uart_tx_stopped(port)) {
        s3c24xx_serial_stop_tx(port);
        goto out;
    }
    /* try and drain the buffer... */
    while (!uart_circ_empty(xmit) && count-- > 0) {
        if (rd_regl(port, S3C2410_UFSTAT) & ourport->info->tx_fifofull)
            break;
        wr_regb(port, S3C2410_UTXH, xmit->buf[xmit->tail]);
        xmit->tail = (xmit->tail + 1) & (UART_XMIT_SIZE - 1);
        port->icount.tx++;
    }
    if (uart_circ_chars_pending(xmit) < WAKEUP_CHARS)
        uart_write_wakeup(port);
    if (uart_circ_empty(xmit))
        s3c24xx_serial_stop_tx(port);
 out:
    return IRQ_HANDLED;
}
```

5.4.3 串口驱动的应用

导航系统的 GPS 设备和 GPRS 设备通过 uart 端口连接到处理器，应用程序通过调用操作系统 API 来实现通信功能，但是系统 API 最终会调用以上实现的驱动函数来实现通信功能。GPS 通信只需要调用 read()系统函数读取串口信息即可，不能完整的检验驱动的正确性，现在我们通过稍微复杂的例子 GPRS 拨号来检验串口的驱动是否正确。

拨号有 2 种方式，一种是使用 wvdial 拨号软件拨号，方便简单，但是程序占用存储空间比较大，一种是使用系统程序 pppd，chat 程序拨号，占用空间小，但是拨号过程复杂，拨号脚本比较难写。嵌入式导航系统要求占用空间小，所以在实现过程中要求我们理清拨号原理，突破难点，写好拨号脚本实现 GPRS 上网功能。

5.4.3.1 拨号脚本

GPRS 拨号需要关键的 2 个脚本文件 gprs_dial 和 gprs_connect。

表 5-1 脚本文件代码

Table 5-1 Script file code

gprs_dial 脚本文件如下所示：	gprs_connect 脚本文件如下所示：
/dev/ttySAC1 115200 noauth noipdefault nodetach user local usepeerdns ipcp-accept-local lcp-echo-failure 30 lcp-echo-interval 4 lock modem defaultroute debug kdebug 4 connect 'chat -v -f/etc/ppp/gprs_connect'	TIMEOUT 5 ECHO ON ABORT '\nBUSY\r' ABORT '\nERROR\r' ABORT '\nNO ANSWER\r' ABORT '\nNO CARRIER\r' ABORT '\nNO DIALTONE\r' ABORT '\nRINGING\r\n\r\nRINGING\r' TIMEOUT 5 " \rAT 'OK-+++\c-OK' AT TIMEOUT 12 OK ATH OK ATE1 OK 'AT+CGDCONT=1,"IP","GZEUNT.GD"' OK 'ATD*99***1#' TIMEOUT 22 CONNECT ""

5.4.3.2 拨号结果

系统使用 pppd 和 chat 程序来进行拨号，如果系统拨号正确，输入 ifconfig 命令将出现 ppp0 的网络接口，能顺利 ping 通网络，比如说百度，就说明拨号已经成功了。测试结果如图 5-9 所示，出现了我们预期的结果，拨号成功。

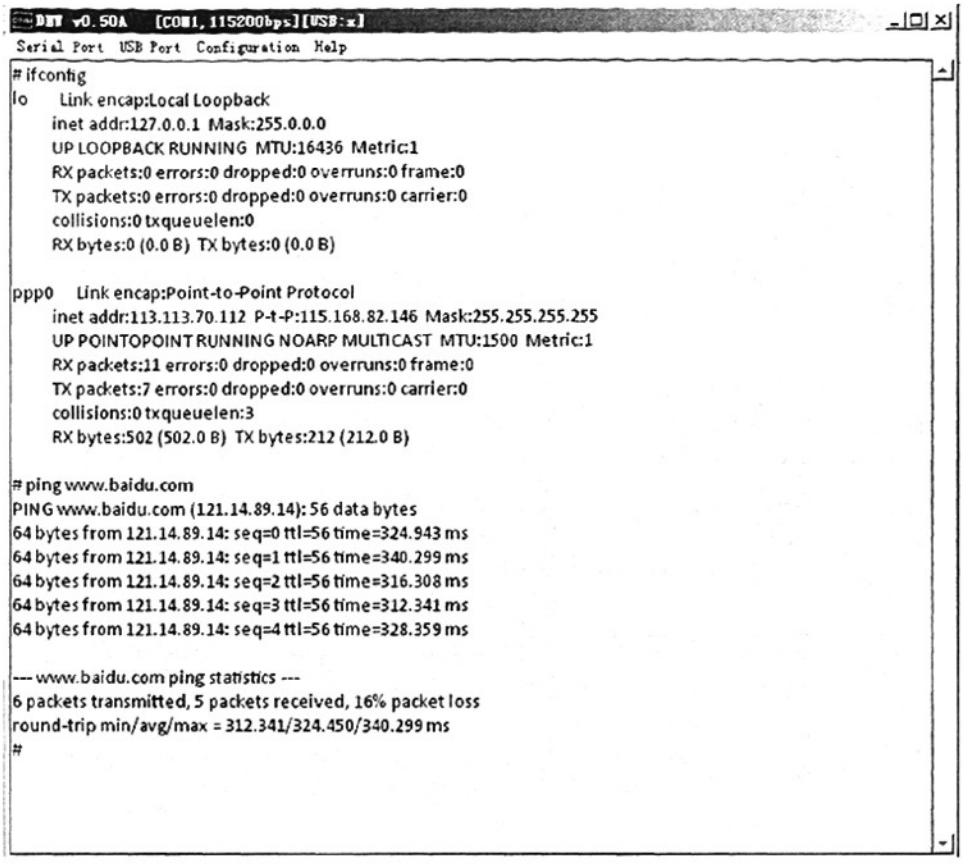


图 5-9 拨号测试结果

Figure 5-9 Dial-up test results

5.5 本章小结

本章介绍了设备驱动程序的一般框架，并针对嵌入式导航外围设备的驱动需求，给出了 USB 主机驱动和触摸屏的驱动的实现方法和实现过程，同时研究了串口驱动，给出了关键驱动代码，最后通过 GPRS 拨号测试了串口驱动的正确性。

第六章 BSP 的优化

6.1 启动速度优化的方法

嵌入式系统中，启动速度是衡量一个产品是否成功的重要的标志，因为，如果手机开机时间需要等待10秒钟以上，客户是不可忍受的。所以在嵌入式系统，特别是消费类电子方面，这方面的优化工作尤为重要。

6.1.1 启动速度优化概述

首先要知道系统哪些耗时，理清了影响启动速度的主要因素，就可针对这些因素进行相应的优化。

1、裁剪内核

既然在启动的最初，内核会被载入内存、解压并执行，那么越是小巧的内核就越是有利于快速的启动，因此，裁剪内核就非常有必要了。关于配置内核的细节，也就是功能的选择则取决于特定的应用。基本的原则就是尽量只保留最基本的功能，能够模块化的功能尽量模块化[44]。

2、减少不必要的检测

为了加快启动速度，就应该减少不必要的检测：

（1） 跳过CPU速度测试

在Linux 启动过程中，内核总是会用自己的方法大致测试一下CPU的速度，结果就是大名鼎鼎的BogoMIPS值。为了测试该值内核会执行很多的短延迟操作，这无疑是要占用启动时间的。因此，可以将一个固定值赋给内核，然后跳过这一步，将这些时间省下来。这个值可以预先测定或根据经验估计。然后将该值赋给loops-per-jify(该变量的位置是init/main.c中的calibrate-delay 0)。

（2） 跳过不存在的设备检测

在Linux 的启动过程中，内核通过检测所有的总线来最终确定系统中的设备。这在很大程度上是没有必要的，所以，在Linux启动时通知内核不去探测不存在的设备将会加快启动。例如，多数机器的ide4都没有设备，那么，就可以传递参数

ide4=noprobe给内核省去检测它们的时间。

（3） 关闭串行终端控制台

大多数的嵌入式开发工程师都是通过串口来调试目标板的，所以，Linux启动时总是会打开串行终端控制台。但也可以关闭这项功能来加快启动速度。要达到这个目的，仅仅需要在启动的时候将参数(quiet)传给内核。

3、优化启动脚本

一旦启动进行到了执行启动脚本这一步，其行为就极具个性化色彩。各种Linux系统在这部分的结构各不相同，所需的运行时间也不尽相同。这些数目繁多，结构复杂的脚本将会读取用户的配置文件并按要求启动各种服务，同时完成一些系统设置。在这一阶段可以采取的措施主要有:

（1）删除当前系统不用的服务

（2）优化脚本组织结构、调整脚本执行顺序

4、采用squanshfs文件系统，代替其他文件系统。

6.1.2 进行进一步优化

系统在之前的内核和文件系统的移植过程中已经使用了很多的优化方法，比如，使用 squanshfs 文件系统，把内核裁剪到最小。但是为了让系统达到最佳的效果，还必须对 BSP 进行进一步的优化。下面是本文根据系统要求，提出的对系统进行进一步优化的措施和方法。

6.1.2.1 预设 lpj 的时间

内核中有一个叫loops_per_jiffy的全局变量，它保存了每0.5个TICK时间范围内CPU可以执行空指令的条数。系统启动时显示如下：

Calibrating delay loop... 199.47 BogoMIPS (lpj=498688)

如果你的系统无法看到此启动信息请在启动参数后面添加loglevel=8

每次启动时，Linux内核都要通过calibrate_delay()函数计算loops_per_jiffy（lpj）的值。这个过程大约需要25jiffies（1个jiffy是2个时钟中断），在嵌入式系统中，大概是250ms。因此，只需要测量一次lpj值，然后在下次启动时，启动参数直接设置lpj=<value>就可以了，S3C2440的Lpj=498688；

启动参数实例：

root=/dev/mtdblock2 load_ramdisk=0 init=/Linuxrc console=ttySAC0 display=sharp640 mem=64M devfs=mount loglevel=8 lpj=498688;

6.1.2.2 驱动程序使用模块动态加载方式

采用并行初始化的思想，并行初始化系统硬件可以加快系统启动时间，所以我们把usb驱动，触摸屏编译成模块，在系统启动后以模块的方式插入内核。

6.1.2.3 关闭串口打印信息

串口打印速度比较慢，关闭串口打印信息，能给系统节约启动时间的30%以上。但是，关闭打印信息之后我们将损失一些信息提示，所以，必须在系统调试完成之后使用。在启动参数后面加上quite即可。

6.2 系统大小优化

1、内核大小优化

目前，在Linux内核的优化方面，有一些组织或个人正在进行研究并取得了一些成果，值得一提的是Linux-Tiny。下载Linux-Tiny的补丁程序并为Linux内核打上补丁之后，在编译内核时只要设置CONFIG_EMBEDDED选项，就可以对内核进行优化了，图6-1显示了采用Linux-Tiny技术前后的内核大小比较。

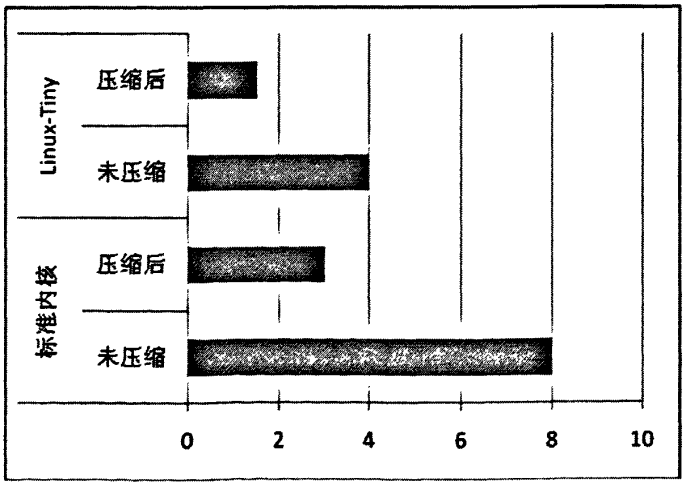


图6-1 内核大小的对比

Figure 6-1 Compare the size of the kernel

2、库文件大小优化

采用uclibc库来编译应用程序。基本的glibc库文件有1.7M，而uclibc库只有500多KB,采用轻量级的uclibc库,可以显著减小应用程序的体积。

6.3 优化结果测试与分析

针对前面所做的优化工作，从启动速度和文件大小二个方面来测试优化结果。

6.3.1 启动速度测试

首先是对Linux启动过程的跟踪和分析，生成详细的启动时间报告。
较为简单可行的方式是通过PrintkTime功能为启动过程的所有内核信息增加时间戳，便于汇总分析。PrintkTime最早为CELF所提供的一个内核补丁，在后来的Kernel 2.6.11版本中正式纳入标准内核。所以大家可能在新版本的内核中直接启用该功能。如果你的Linux内核因为某些原因不能更新为2.6.11之后的版本，那么可以参考CELF提供的方法修改或直接下载它们提供的补丁：
http://tree.ceLinuxforum.org/CelfPubWiki/PrintkTimes 开启PrintkTime功能的方法很简单，只需在内核启动参数中增加“time”即可。当然，你也可以选择在编译内核时直接指定“Kernel hacking”中的“Show timing information on printks”来强制每次启动均为内核信息增加时间戳。这一种方式还有另一个好处：你可以得到内核在解析启动参数前所有信息的时间。因此，本系统选择后一种方式。
当完成上述配置后，重新启动Linux，然后通过以下命令将内核启动信息输出到文件： dmesg -s 131072 > ktime 然后利用一个脚本“show_delta”（位于Linux源码的scripts文件夹下）将上述输出的文件转换为时间增量显示格式：show_delta ktime > dtime 这样，就得到了一份关于Linux启动时间消耗的详细报告。

在报告中的时间增量和内核信息之间没有必然的对应关系，真正的时间消耗是从内核源码入手分析的。因为时间增量只是两次调用printk之间的时间差值。通常来说，内核启动过程中在完成一些耗时的任务，如创建hash索引、probe硬件设备等操作后会通过printk将结果打印出来，这种情况下，时间增量往往反映的是信息对应过程的耗时；但有些时候，内核是在调用printk输出信息后才开始相应的过程，那么报告中内核信息相应过程的时间消耗对应的是其下一行的时间增量；还有一些时候，时间消耗在了两次内核信息输出之间的某个不确定的时段，这样时

间增量可能就完全无法通过内核信息反应出来了。

所以，为了准确判断真正的时间消耗，需要结合内核源码进行分析，但是现在主要是测试内核启动的总的时间，所以，这样的误差不会影响到总体启动时间的度量。

优化之前系统启动时间情况（串口输出的信息，因信息过长，这里只截取开头与结尾）如图6-2，启动时间为6.355秒。

```
[21474536.480000] Linux version 2.6.12-h1940 (root@localhost.localdomain) (gcc
version 3.4.1) #3 Sun Mar 29 11:08:41 CST 2009
[21474536.480000] CPU: ARM920Tid(wb)
[21474536.480000] S3C2440: core 400.000 MHz, memory 100.000 MHz, peripheral
50.000 MHz
[21474536.480000] Kernel command line: root=/dev/mtdblock2  load_ramdisk=0
init=/linuxrc console=ttySAC0 display=sam320 mem=64M devfs=mount  loglevel=8
[21474536.515000] Memory: 61952KB available (2307K code, 459K data, 108K init)
[21474536.520000] Calibrating delay loop... 199.47 BogoMIPS (lpj=498688)
[21474536.650000] S3C2440: Clock Support, UPLL 48.000 MHz
[21474536.655000] SCSI subsystem initialized
[21474536.655000] usbcore: registered new driver usbfs
[21474536.660000] usbcore: registered new driver hub
[21474536.680000] S3C2410 DMA Driver, (c) 2003-2004 Simtec Electronics
[21474536.680000] DMA channel 0 at c4800000, irq 33
[21474536.700000] NetWinder Floating Point Emulator V0.97 (double precision)
[21474536.730000] devfs: 2004-01-31 Richard Gooch (rgooch@atnf.csiro.au)
[21474536.730000] devfs: boot_options: 0x1
[21474536.740000] yaffs Mar 29 2009 11:02:20 Installing.
[21474536.745000] Initializing Cryptographic API
[21474536.750000] --- s3c2410fb init ---!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
[21474536.760000] pxafb_probe start!!!!!!!!!!!!!!!!!!!!!!!!!!!
[21474537.610000] RAMDISK driver initialized: 16 RAM disks of 4096K size 1024
blocksize
[21474537.720000] NAND device: Manufacturer ID: 0xec, Chip ID: 0x76 (Samsung
NAND 64MiB
.....
[21474541.600000] VFS: Mounted root (squanshfs filesystem) readonly.
[21474541.650000] Mounted devfs on /dev
[21474542.835000] selected clock c0279208 (pclk) quot 26, calc 115740
```

图6-2 优化前启动情况

Figure 6-2 Optimization of pre-optimize

没有关闭串口打印信息，优化之后系统启动时间情况如图6-3，启动时间为4.62秒，如果关闭串口打印，将要节省30%时间，经测定最终启动时间为3.56秒。

```
[21474536.480000] Linux version 2.6.12-h1940 (root@localhost.localdomain) (gcc
version 3.4.1) #3 Sun Mar 29 11:08:41 CST 2009
[21474536.480000] CPU: ARM920Tid(wb)
[21474536.480000] S3C2440: core 400.000 MHz, memory 100.000 MHz, peripheral
50.000 MHz
[21474536.480000] Kernel command line: root=/dev/mtdblock2  load_ramdisk=0
init=/linuxrc console=ttySAC0 display=sam320 mem=64M devfs=mount  loglevel=8
[21474536.510000] Memory: 61952KB available (2307K code, 459K data, 108K init)
[21474536.515000] Calibrating delay loop (skipped)... 199.47 BogoMIPS preset
[21474536.550000] S3C2440: Clock Support, UPLL 48.000 MHz
[21474536.565000] S3C2410 DMA Driver, (c) 2003-2004 Simtec Electronics
[21474536.565000] DMA channel 0 at c4800000, irq 33
[21474536.585000] NetWinder Floating Point Emulator V0.97 (double precision)
[21474536.600000] devfs: 2004-01-31 Richard Gooch (rgooch@atnf.csiro.au)
[21474536.600000] devfs: boot_options: 0x1
[21474536.605000] yaffs Mar 29 2009 11:02:20 Installing.
[21474536.615000] Initializing Cryptographic API
[21474536.965000] RAMDISK driver initialized: 16 RAM disks of 4096K size 1024
blocksize
[21474537.990000] NAND device: Manufacturer ID: 0xec, Chip ID: 0x76 (Samsung
NAND 64MiB
.....
[21474539.865000] VFS: Mounted root (squanshfs filesystem) readonly.
[21474539.915000] Mounted devfs on /dev
[21474541.100000] selected clock c0279208 (pclk) quot 26, calc 115740
```

图6-3 优化后系统启动情况

Figure 6-3 bootup Information of the optimized system

从结果可以分析得到

表 6-1 系统启动速度对比

Table 6-1 the speed of system startup Comparision

	开始时间	结束时间	所用时间 未加quite参数	最后所用时间
优化前	2174536.480000	21474542.835000	6.355	6.355
优化后	2174536.480000	21474541.100000	4.620	3.56

经过优化之后启动速度快了近3秒。对于嵌入式系统来说，这是一个很大的飞跃。

6.3.2 文件大小优化测试

文件大小优化测试方法就比较简单了，不需要特别的方法，只需要通过ls –sh

或者du –sh命令就可以知道文件的大小了。

表6-2 文件大小对比

Table 6-2 File Size Comparison

	未压缩	已压缩
优化前	8M	3.4M
优化后	5M	2.1M

尚未加入应用程序的情况下达到30%的优化成果，可见优化结果是令人满意的。

6.4 本章小结

本章结合本嵌入式导航系统的实际要求，研究了启动速度优化和系统文件大小优化的方法，与没有优化的BSP相比，本BSP具有启动速度快，占用存储空间少的优点，有效地降低了系统成本。从优化前后的结果对比得知，优化取得了令人比较满意的效果。

第七章 优化 BSP 在嵌入式智能交通导航系统的应用

7.1 智能交通导航系统需求

嵌入式智能交通导航系统是一个复杂的系统，要求界面友好，操作方便，响应速度快，地图显示连贯， GPRS 实时通信。整个系统主要实现四大部分的功能，GPS 信息管理，电子地图管理，导航管理，GPRS 在线监控。

对比传统的导航系统，嵌入式智能交通导航系统具有以下几种新的需求：

1、要运行复杂的智能导航算法，必须要有运算速度快的 CPU 和大容量的内存。本系统路径规划算法使用改进的 A*算法，它的优点是能得到真正的优化路径，缺点是在计算时要消耗大量的系统资源，时间需要比较长，因此需要高速 CPU 和大容量内存。

2、要实现友好的人机界面和流畅的地图显示，必须要有强大的图形库支持和触摸屏支持。本系统使用触摸屏代替传统的键盘，实现更人性化的人机操作界面。

3、可写的文件系统。系统需要记录一些用户的使用习惯信息，系统配置信息，存储监控指令，需要可写的文件系统。

4、通信必须实时在线。系统还需要在线监控功能，监控端要实时的得到手持终端的位置信息，并且要能向手持终端发送指令，GPRS 等类型的通信方式能满足系统要求。

7.2 智能交通导航软件功能介绍

导航应用程序包括四大，电子地图显示模块，GPS接收模块，GPRS通信模块以及导航管理模块。

电子地图显示模块包括以下几个部分，系统界面与管理、地图数据管理、坐标转换、电子地图的显示。系统界面与管理实现系统主界面和操作菜单，通过调用其它模块实现地图漫游、自动对中、信息查询等功能。地图数据管理负责读取电子地图文件，实现地图查询功能。坐标转换：地理坐标、墨卡托投影坐标和屏

幕坐标之间的相互转换。地图显示根据管理模块给出的中心点位置及显示范围的参数，确定需要绘图的网格.根据网格索引，快速完成绘制地图的功能。考虑到地图漫游通常是连续的，使用了位图缓存来提高响应速度，每次画图的时候，先在缓存中找，找不到则扩大范围画图，并保存到位图缓存。

GPS 接收模块：负责定时读取GPS 接收仪的数据，解析出地理经纬度坐标、时间、速度与方向等信息。外置的GPS 接收仪通过RS232 连接到系统，GPS 接收仪会定时往系统发送符合NMEA标准的数据。该模块只需定时读取RS232 串口数据， 根据NMEA协议解析出地理经纬度坐标、时间、速度与方向的信息即可。

GPRS通信模块：负责向控制中心定时发送手持终端所在的位置，速度，方向等信息。

导航管理模块：根据电子地图中的道路拓扑信息结合GPS和GPRS接收到的信息，在全局已知的环境下做路径规划，并完成路线设置、路线偏移报警、保存运行轨迹等管理功能。路径规划采用改进的A*算法,路线设置是根据规划结果形成一条路线，并对该路线设置一个偏移极限。

7.3 基于优化 BSP 的嵌入式智能交通导航系统设计要点

基于优化 BSP 的嵌入式把智能导航系统是分为三个模块来实现的。

（1）通信模块，通信模块 BENQ M23g 通过串口交互信息，实现实时在线通信。BSP 已经提供了 3 个串口的驱动程序，在第五章我们研究过串口驱动，同时给出了拨号脚本，应用系统通过 GPRS 网络把数据发往监控端的数据库，同时从 GPRS 网络接收最新的指令。应用程序通过底层提供的 API，设计一个守护进程，随时监控网络数据。

（2）导航数据接收，GPS 数据接收模块周期性的往串口发送 GPS 定位数据，应用程序通过调用 linux API 读取串口信息，并解析信息，把信息提供给处理模块。BSP 的串口驱动很好的提供了底层的支持。

（3）数据处理模块，这个是系统的核心处理模块，包括地图的显示，数据存储，导航路径规划算法等，这些功能都必须由 BSP 提供底层的支持来实现，具体可以从以下几个方面来体现：

BSP 对显示模块的支持：电子地图显示通过 lcd 显示，输入通过触摸屏控制输入。BSP 包已经提供了相关的底层处理程序。应用系统需要移植图形库支持，这里选择 QTE 图形库，通过图形库的支持，设计人性化得交互界面，和流畅的地图显示。

BSP 对数据存储的支持：BSP 除了提供存储空间给应用软件外，还独立提供 52M 的空间供用户存储电子地图信息，52M 的空间应该可以存储单个城市的电子地图信息，如果用户需要在全国各地跑，也不用担心系统空间不够，因为系统通过 USBhost 接口支持外扩 USB 设备，通过 U 盘存储电子地图信息，导航应用软件可以优先读取 U 盘中的电子地图信息来进行导航。应用程序只需要考虑读取数据就可以了。不用担心存储空间扩展的问题

导航算法主要是由应用程序完成，系统已经提供 400MHZ 运行能力的 CPU，大大满足系统的需求了。

BSP 对导航软件的升级和导航地图的更新的支持，导航系统软件在未来的发生更新换代的几率很大，系统同时提供 USB 接口留个用户升级导航软件。用户还可以通过 USB 接口更新自己的地图信息。应用程序在人机界面上提供给用户一个更新接口就可以了。

通过本文研究的 BSP，智能导航系统已经有了一个稳定的运行基础平台，应用程序开发人员只需要一心一意的处理应用软件的开发工作。

7.4 智能导航系统的实现结果

应用程序已经在本 BSP 上成功实现。系统采用 qte 图形库开发图形界面，全部采用触摸屏控制方式，人机交互界面非常友好。从大学城广东工业大学图书馆到地铁 4 号线大学城南站导航手持终端导航路线截图，如图 7-1 所示。

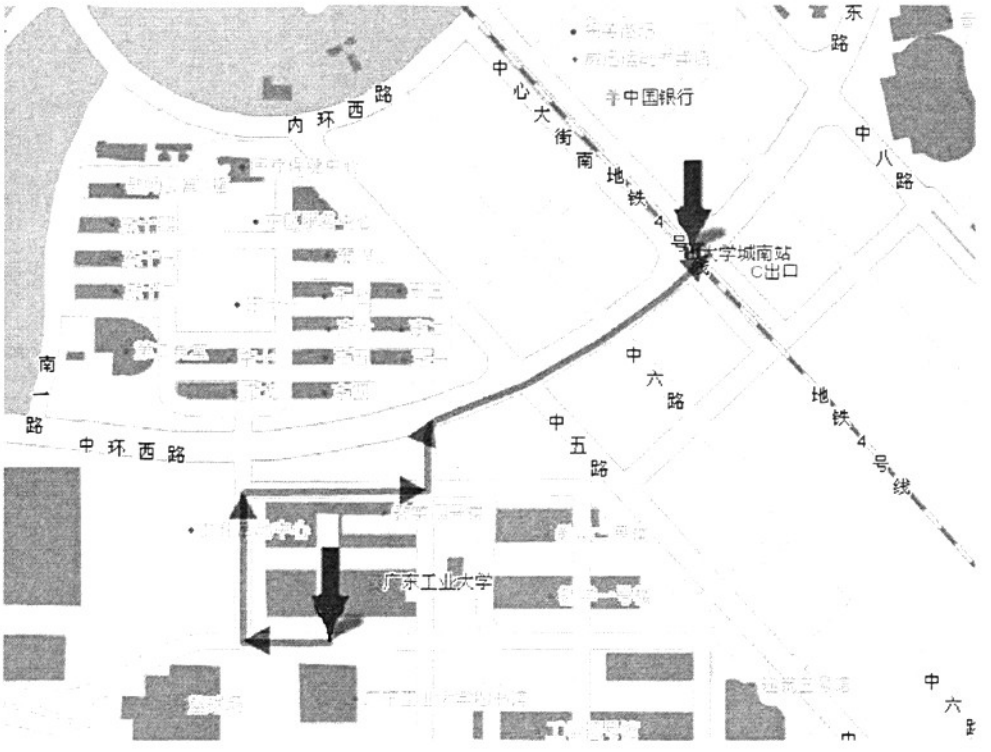


图7-1广东工业大学图书馆至大学城南站

Figure 7-1 Guangdong University of Technology Library to the University of South Station

经过实验对比，处理速度比 S3C2410 嵌入式平台有了很大的提高，系统响应速度快，地图显示流畅，规划出来的航线比原来的嵌入式平台在精度上也有了很大的改进。如图 7-2 所示。蓝色线条即为所规划出来的路线。

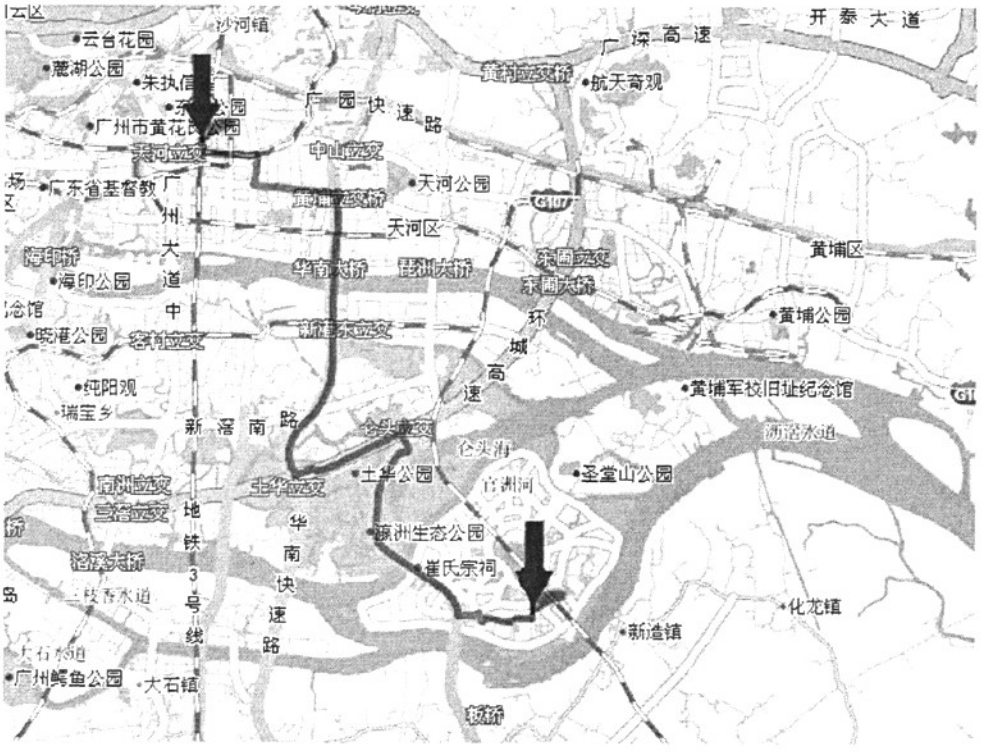


图 7-2 广东工业大学至天河体育中心

Figure 7-2 Guangdong University of Technology to Tianhe Sports Center

7.5 本章小结

本章用一个实际的例子对嵌入式 BSP 进行了深入的验证，在此平台上实现了智能交通导航系统。与其他平台相比，本 BSP 上实现的导航系统有以下几个特点：

（1）经过启动优化，系统启动时间速度快，

（2）响应速度快，采用最新的 ARM 处理器 S3C2440，从硬件层面上保证了系统的响应速度，克服了原有嵌入式导航系统运行复杂算法响应速度慢的缺陷。

（3）人机交互方便界面友好，用触摸屏代替传统的键盘，和强大的图形库（QTE）的支持下，人机交互便利，操作界面友好。

（4）数据交换速度快、实时在线。采用 GPRS 数据传输，导航系统能与控制中心进行实时的交换数据。

在本 BSP 上实现的智能交通导航系统，启动和数据更新速度较快，运行稳定，从而充分验证了本 BSP 包的设计达到了预期的效果。

结 束 语

本文针对嵌入式导航系统的应用需求，完成了板级支持包软件BSP的设计。课题完成了针对嵌入式Linux2.6.14版本的内核的引导程序设计，分析了引导程序的结构，功能，以及boot loader实现的关键点并且给出了示例代码。在分析内核源码的基础上，针对Linux2.6.14版本的内核，完成了针对嵌入式导航系统硬件相关代码的定制，实现嵌入式Linux在目标板上的移植。并且根据导航系统外扩设备的要求，从分析Linux下设备驱动设计的机制入手，分别就USB驱动，触摸屏驱动和串口驱动的设计关键进行了深入的分析，在此基础上给出了关键的数据结构和驱动接口，以及驱动框图，为驱动开发提供了一个较为完整的设计过程。

本文的主要成果为：

1) 采用SAMSUNG公司的32位S3C2440处理器作为导航系统的硬件平台 ，在此平台上，实现了引导代码boot loader的设计与实现，使它在目标板上电后完成硬件平台的初始化，完成了boot loader的下载功能与引导内核启动功能，并且提供了人性化的操作界面。
2) 针对S3C2440处理器以及嵌入式导航系统的硬件需求，完成了嵌入式Linux26.14内核与squashfs、yaffs2、tmpfs文件系统的移植。本文配置内核使其支持压缩的文件系统。squashfs映像，并通过传递kenrel command参数，引导根文件系统的加载。重点完成了内核的硬件相关部分部分代码的定制和根文件系统的定制过程。
3) 针对导航系统连接的外围硬件接口，设计了USB驱动程序，使得系统能够顺利读写U盘。设计了触摸屏的驱动程序，实现用户与系统的交互目标。
4) 针对嵌入式导航系统，提出了多项优化方法，通过对BSP的整体的优化，系统启动时间比优化前要快，BSP文件大小比优化前要小，有效地提高了系统性能和减少了存储空间的消耗。

本文重点完成了板级支持包软件的设计，在此基础上，可以从以下方面进行进一步研究。

在驱动程序设计过程中，由于典型设备驱动有许多标准的流程，可以考虑设

计一个驱动开发平台，根据不同的设备类型，自动生成基本的驱动框架，例如字符型设备可以有一个字符型设备驱动的框架，完成字符型设备的注册，卸载等必须完成的代码，而只留下I/O管理，中断服务程序处理等具有硬件特定的代码作为驱动开发的重点，这样可以明显减少驱动开发的周期，而且将驱动开发人员的精力主要集中在实现硬件特性的目标上。

Linux操作系统是个多任务分时操作系统，它在实时性方面一直存在着缺陷，在以后的优化过程中，可以考虑在Linux操作系统源代码基础上添加软实时功能，像monta vista实时操作系统一样，提供软实时能力。

参 考 文 献

[01] 龚真春,嵌入式 GPS/MIMS 组合导航系统的设计与应用[J],计算机工程与应用，2005（108）.

[02] 胡小平,自主导航理论与应用[M],国防科技大学出版社.

[03] 胡力,陈耀武,汪乐宇,基于 GPS 和电子海图的嵌入式船舶导航系统设计[J]，电子技术应用.Vol.14, No.6, 7-9,2005

[04] 蔡仲伦,艾长胜,孙选等,基于 GPS 的车辆导航及 GPRS 网络监视[J]. 电子设计应用,Vol.4,2006

[05] http://www.embed.org/.

[06] http://www.zdnet.com.cn/developer/webdevelop/story/0,3800067013,3915699600.htm, David Brenan，详细定义嵌入式系统.

[07] 吴朝晖，纵谈嵌入式技术[J]，微电脑世界，2005（49）

[08] 何丽，嵌入式 Linux：挑战与发展共存[J]，微电脑世界，2001（1）

[09] htpp://www-128.ibm.com/developerworks/library/l-solar/#main，Guid to porting form solaris to Linux。

[10] http://net.pku.edu.cn/~yhf/lyceum/LinuxK/sources/sources.html，Linux 核心资源。

[11] 葛小明，2004 年嵌入式 Linux 市场分析[J]，开发系统世界，2004（4）

[12] Daniel P.Bovet & Macro Cesati，“Understanding the Linux Kernel”，2004，6th edition。

[13] 孙天泽，袁文菊，张海峰等，嵌入式设计及 Linux 驱动开发指南[M]. 北京：电子工业出版社，2005

[14] 夏林峰，嵌入式系统的 BSP 软件包设计[J]，IC 与元器件，2003（12）

[15] http://dev.yesky.com/87/2618087.shtml，深入浅出 Linux 设备驱动编程之引言。

[16] 吴珊，航海导航系统的板级支持包软件设计[J]，浙江大学硕士学位论文。

[17] 丁晓波，基于嵌入式 Linux 的 BSP 技术研究[J]，电子科技大学，2005（3）

[18] 孙彦景，马小平，实施操作系统中的板级支持包 BSP[J]，单片机与嵌入式系统，2002（4）

[19] 孙天泽，袁文菊，张海峰。 嵌入式设计及 Linux 驱动开发指南：基于 ARM9 处理器[M].北京：电子工业出版社，2005.80284，95298

[20] 马学文，嵌入式系统中的 Boot loader 的设计与实现[J].计算机工程，2005（7）：96-97。

[21] Karim Yaghmour . Building Embedded Linux Systems [M].北京：中国电力出版社，2004. 2552260

[22] http://www.Linuxsir.org/bbs/thread107039.html, 嵌入式 BootLoader 技术内幕.

[23] 赵星寒 ARM开发工具ADS原理与应用[M]. 北京.北京航天航空大学出版社.2006

[24] Samsung. S3C2440A Datasheet[Z].2004

[25] http://Linux.21ds.net/2002/08/30/260e5abb82ce840f7fc54bc38d0b8846/，Linux 文件系统基础。

[26] R Love, 莉君，康华，张波等译, Linux 内核设计与实[M]. 北京：机械工业出版社，2006

[27] http://aijuns.blogchian.com/3226056.html，Linux 系统移植。

[28] ftp://ftp.gnu.org/gnu

[29] Christopher Hanllinan Embedded Linux Primer[M] Pearson Education 2008

[30] Moshe Bar，“Linux 文件系统”，清华大学出版社，2003

[31] 田家林，陈利学，寇向辉。 LINUX嵌入式操作系统在ARM的移植[J]。 微计算机信息 Vol.23 No.4-2,2007

[32] 王艳春，崔洪启等，Linux 虚拟文件系统分析[J]，长春理工大学学报，2005(1):59-61

[33] Daniel P.Bovet , Macro Cesati，Understanding the Linux Kernel[M]，2004，6th edition

[34] http://free-electrons.com/docs/optimizations/ ，embedded_Linux_optimizations

[35] http://squashfs.sourceforge.net/,，SQUASHFS

[36] http://hi.baidu.com/zmingliu/blog/item/db48f8fad303689159ee908e.html，YAFFS2 概述

[37] http://all.zcom.com/article/62/a23591/ 在 Linux 中实现大容量 NAND Flash 的 YAFFS2 文件系统

[38] http://www.busybox.net/downloads/README ，README

[39] http://eLinux.org/Printk_Times ， Printk_Times

[40] Alessandro Rubini & Jonathan Corbet，“Linux Device Drivers”，2002(11)

[41] http://www.everythingusb.com/timeline.html，The USB Vision: 10 Years Later

[42] 宋宝华，Linux 设备驱动开发详解释[M]。北京.人民邮电出版社。2008

[43] http://www.ebdev.com/EOS/YAFFS-FileSystem.pdf， Linux MTD，Yaffs howto

[44] Andrew N.Sloss Dominic Symes著， 沈建华译。 ARM嵌入式系统开发—软件设计与优化[M] ，北京航空航天大学出版社 。2005

[45] 殷惠莉，刘少君，黄道平。基于uCLinux触摸屏的设计[J]. 电子工程师. 2004(2)

攻读学位期间发表论文

[1] Jianqi Liu, Bi Zeng, Xiuwen Yin, Biliang Zhong, Xiuzhang Zheng,《Designation and Realization of Ship Navigation System Embedded Platform Based on ARM》, the fifth IEEE International Symposium on Embedded Computing , 2008.10.

[2] 刘建圻，曾碧，郑秀璋，《基于 RBAC 权限管理模型的改进与应用》，计算机应用，2008.9

[3] 刘建圻，曾碧，郑秀璋，钟碧良，《基于 S3C2440 的嵌入式导航平台的设计与实现》自动化与信息工程，2008.7

[4] 易红光，林伟，刘建圻，《基于物理层隔离的单向文件轮渡系统的设计及应用》，计算机安全，2009.1

致 谢

衷心感谢曾碧教授。在三年的学习阶段，始终得到曾老师的热情关怀和精心指导。一直以来，曾老师在学习、工作、生活等多方面给予了我诸多帮助和关怀，为我创造了一个良好的学习环境以及许多难得的锻炼机会。曾老师渊博的学识、丰富的经验、严谨求实的治学态度、忘我的工作精神使我受益终身。研究生学业即将完成之际，谨向曾老师致以最衷心的谢意。

衷心感谢余永权教授、林伟教授对我学业的关心与支持，为我完成学业提供了一个良好的工作环境。

.感谢实验室苏振文、彭辉、徐敏霖、韩洁琼等师兄师姐，在实验室的工作与学习过程中，他们的经验使我受益匪浅。

感谢我的同窗郑秀璋、尹秀文等，在实际的项目中，他们始终给予我很大的帮助。

感谢我的师弟方坤涛、曾毅、徐以山等，感谢他们对我工作的支持。

最后，我要特别感谢我的父母和家人，他们对我始终如一的支持和关怀使我能够顺利完成学业。还要感谢许多关心我的朋友，感谢他们的无私帮助和大力支持。

刘建圻

2009年5月于广东工业大学