ARM Linux 中断系统移植研究

陈何杰, 郑灵翔*
(厦门大学信息科学与技术学院, 福建 厦门 361005)

摘要: 基于 ARM 的嵌入式 Linux 移植与应用是当今嵌入式领域的热点与难点, Linux 中断的移植在整个嵌入式 Linux 的移植应用中占有重要地位. 针对嵌入式 Linux 的中断系统移植进行了深入的研究, 通过分析 Linux 源代码, 研究了 Linux 中断系统的结构和移植的软硬件接口, 并以 MP2530F 开发板为例, 介绍了 Linux 中断的移植方法, 分析了移植过程中面临的问题及其解决办法. 由于在设计中使用了面向对象的方法与两种设计模式, 使得 Linux 中断系统具有良好的架构设计, 并有效地屏蔽了底层硬件实现的复杂性, 提高了 Linux 内核的可移植性. 对于理解 Linux 内核的中断系统, 以及嵌入式 Linux 的中断移植均有一定的参考价值.

关键词: ARM Linux; 中断; 面向对象; 设计模式
中图分类号: TP 316.89 文献标识码: A 文章编号: 0438-0479(2010)03-0339-05

在以微控制器(MCU)和微处理器(MPU)为核心的嵌入式系统中, ARM 核心的各种 MCU/MPU 占据了半壁江山. 与此同时, 嵌入式 Linux 以其开源的特性以及低廉的开发成本等因素位居嵌入式操作系统榜首[1]. 基于 ARM 的嵌入式 Linux 开发成为当今嵌入式领域炙手可热的话题.

嵌入式开发中, 嵌入式操作系统的移植和开发一直是一个比较困难的项目[1]. 由于嵌入式产品上搭载的芯片、外围设备各不相同, 因而需要根据硬件环境进行嵌入式操作系统的移植. 这其中, 中断处理系统的移植是嵌入式操作系统移植的一个重要组成部分.

嵌入式 Linux 的中断处理实现依赖于处理器、中断控制器的类型、体系结构的设计及机器本身[2]. Linux 中断移植不仅需要充分了解开发板的硬件, 同时还需要对 Linux 内核中断系统的结构与运作有较深入的理解. 但是 Linux 内核代码极为庞大和复杂, 同时目前相关资料较缺乏, 这给 Linux 的中断移植造成了很大的困难. 为此, 本文将基于 ARM 处理器, 对嵌入式 Linux(版本号 2.6.20) 的中断系统进行分析. 在此基础上, 以韩国美智公司的 MP2530F 开发板为例, 介绍中断的移植与应用.

1 ARM Linux 内核的中断系统分析

1.1 中断系统的结构

Linux 内核是一个庞大而复杂的操作系统的核心, 不过尽管庞大, 但是却采用子系统和分层的概念很好地进行了组织. 它主要由 5 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. 中断系统是进程调度子系统中的重要组成部分, 它的源代码可以在内核源码的 kernel 目录中找到, 同时内核源码的 arch 目录中可以找到其依赖于体系结构的源代码.

Linux2.6 内核的设计采用了面向对象和设计模式的思想, 同时采用 C 语言来实现, 在保持代码高效运行的同时又不失可移植性和可扩展性, 并且易于维护. Linux2.6 内核的中断系统可以用图 1 来表示[3].

图 1 中的各抽象类均来自内核代码, 主要根据各个重要结构体抽象而成. 该图的中心是中断描述数组 irq_desc, 数组元素对应于各个中断号, 分别描述相应中断号的信息与处理方式. 抽象类 irq_desc 主要由 irq_chip、irq_flow_handler 和 irqaction 3 个抽象类聚合而成:

1) irq_chip 包含了中断硬件的相关操作方法, 图中 kernel_defined_chip 表示 Linux 内核源码中自带的常用中断控制器(例如 8259A); user_defined_chip 表示用户根据所使用的硬件添加的中断控制芯片.

2) irq_flow_handler 定义了内核中各种类型的中断处理方式, 图中 kernel_defined_handler 表示内核定义的几种中断处理方式, 例如 handle_level_irq 和 handle_edge_irq 等; user_defined_handler 表示用户可

收稿日期: 2009-08-21
基金项目: 福建省自然科学基金(2009J01306)
* 通讯作者: lxzheng@xmu.edu.cn

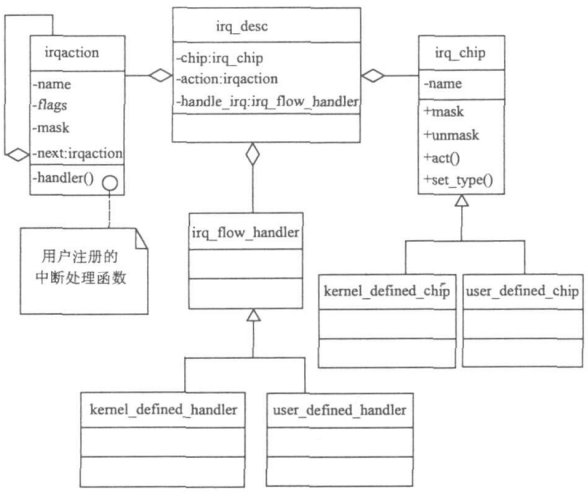

图 1 Linux 内核中断抽象类图

Fig. 1 Abstract class diagram of Linux interrupt system

以根据需要添加中断处理方式.

3) irqaction 包含了待处理的中断处理函数,它们连接在一个链表上,该链表以中断号作为标志.

为了更好地封装代码,屏蔽中断硬件的细节,便于用户进行开发,Linux2.6 内核的中断部分主要借鉴了两种设计模式的思想,它们分别是策略(strategy)模式和职责链(chain of responsibility)模式。

策略模式的思想主要应用在 irq_chip 及 irq_flow_handler 上. 该模式的意图在于定义一系列的算法,把它们一个个封装起来,并且使它们可以相互替换[4]. irq_chip 和 irq_flow_handler 的结构恰好满足策略模式应用的几个条件:

1) 许多相关的类仅仅是行为有异:内核及用户定义的 irq_chip 的结构是一致的,只是根据硬件不同,在控制的代码上有差异;irq_flow_handler 则是根据中断类型的不同,在处理方式上稍有差别.

2) 需要隐藏算法相关的数据: irq_chip 和 irq_flow_handler 分别将硬件相关和中断类型相关的数据隐藏起来,使得它们对内核透明,降低了代码耦合性与处理的复杂性.

策略模式的使用降低了内核代码之间的关联与耦合,以及代码封装的复杂性. 该模式的采用一方面屏蔽了底层实现的复杂性,另一方面为开发人员提供了一个很好的接口,便于嵌入式移植应用. 嵌入式 Linux 的中断移植只需要根据 irq_chip 和 irq_flow_handler 的规范,编写具体硬件的控制代码和中断处理代码,而不需要修改内核的其他部分.

由于中断号非常有限,Linux 需要对中断号进行共享. Linux 内核在中断共享上应用了职责链模式的思想,构造了职责链 irqaction. 职责链模式的思想是将对象连成一条链,并沿着这条链传递请求,直到有一个对象处理请求为止[4]. 它的应用条件跟中断共享处理的需要相吻合:

1) 有多个的对象可以处理一个请求,哪个对象处理该请求运行时刻自动确定:共享的中断被触发时,内核并不知道具体是由哪个设备触发,这一点交给职责链上的对象在运行时自行判断.

2) 可处理一个请求的对象集合需要被动态指定:用户可以随时申请一个共享的中断,即可以动态地增加某一共享中断上的设备.

通过使用职责链模式,用户申请中断时,内核将中断处理函数挂到职责链上;对应中断触发时,内核检查链上的各个中断处理函数,进行相应处理. 如此实现了中断共享,充分利用了有限的中断资源,降低了代码的耦合. 职责链模式的使用也方便了中断系统的移植,开发人员可以使用丰富的中断资源,同时不需要考虑中断共享的处理.

1.2 中断系统的运行

1.2.1 中断初始化

Linux 内核在初始化阶段完成了对页式虚存管理的初始化后,便调用 trap_init() 和 init_IRQ() 两个函数进行中断机制的初始化. 其中 trap_init() 主要是对一些系统保留的中断向量的初始化,而 init_IRQ() 则主要是用于外设的中断[5]. init_IRQ() 位于 arch\arm\kernel\irq.c,它完成中断描述数组 irq_desc 的初始化,主要初始化中断的状态信息. 之后内核跳转到硬件相关的初始化函数 init_arch_irq,该函数在编译时根据实际体系结构被指定(类似于编译时多态),它将初始化中断控制硬件,并注册硬件控制函数和中断处理函数,即对 desc 的 chip 和 handle_irq 进行赋值.

1.2.2 中断申请

用户通过 request_irq 操作来申请中断. request_irq 的原型是:

int request_irq(unsigned int irq, irq_handler_t handler, unsigned long irqflags, const char* devname, void* dev_id).

它的各参数含义如下[3,6]:

1) unsigned int irq:所申请的中断号.

2) irq_handler_t handler:注册的中断服务程序,request_irq 操作将它设置为用户定义的中断处理函数.

3) unsigned long irqflags:中断类型标志,它可以

设为以下值:

- IRQF_SHARED: 允许其他设备共享该中断号, 相当于 2.4 内核的 SA_SHIRQ.
- IRQF_DISABLED: 执行 ISR 时关闭本地中断, 相当于 2.4 内核的 SA_INTERRUPT.
- IRQF_SAMPLE_RANDOM: 告诉内核本中断源可以作为随机数生成器的熵池, 相当于 2.4 内核的 SA_SAMPLE_RANDOM.
- const char * devname: 发出中断请求的设备名
- void * dev_id: 将 irqflags 设置为SA_SHIRQ 时, 必须指定一个唯一的 dev_id 作为该共享中断号上的处理程序的标识, 即 irqaction 职责链上的标识符.

内核响应该中断申请, 创建一个 irqaction 类型的 action 结构体, 根据用户的参数设置该 action, 包括将用户的中断服务程序 handler 赋给 action-> handler. 最后将该 action 挂到该中断号的 irqaction 职责链上, 完成了中断的申请.

1.2.3　中断执行

中断的执行过程可以用图 2 表示:

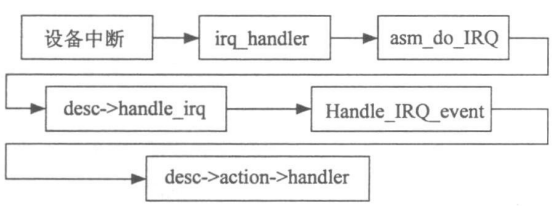

图 2　Linux 中断执行过程

Fig. 2　The execution of Linux interrupt

设备发出中断请求后, CPU 根据中断向量表进行跳转, 调用宏 irq_handler. 该宏获取中断号, 并以中断号和寄存器信息为参数, 调用中断通用处理函数 asm_do_IRQ. asm_do_IRQ 负责处理所有硬件 IRQ, 它通过 desc_handle_irq 函数调用 dese-> handle_irq. 该 handle_irq 是对应中断号的中断处理函数, 由它调用用户注册的中断服务程序(图 2 的 dese-> action-> handler)完成中断处理, 这个过程一般要经过以下几个步骤:

1) 给 dese-> lock 上锁, 防止别的 cpu 访问该中断例程.

2) 发送 AC 应答给中断控制器, 表示响应了中断.

3) 根据需要设置、清除中断标志 dese-> status, 或者禁用该中断.

4) 给 dese-> lock 解锁, 调用 handle_IRQ_event 函数处理中断.

5) 通过 handle_IRQ_event 函数调用 dese-> action-> handler, 执行用户注册的中断服务程序(通过 request_irq 注册).

6) 中断服务程序执行完毕, handle_IRQ_event 返回.

7) 给 dese-> lock 上锁, 根据需要设置 dese-> status. 如果前面禁用了该中断, 则重新启用.

8) 给 dese-> lock 解锁, 中断处理完毕.

1.2.4　中断释放

中断使用完毕后, 可以通过 free_irq 来释放. 它的函数原型是:

void free_irq(unsigned int irq, void * dev_id).

两个参数含义和 request_irq 一致. 释放共享中断时, 将根据 dev_id 标志释放 irqaction 职责链上的相应对象.

内核接到中断释放请求, 遍历该中断号的 irqaction 职责链, 找到对应 dev_id 的 action 对象, 将其从链上移除并释放. 与此同时, 内核还调用注册的 irq_chip 信息关闭该硬件中断.

2　嵌入式 Linux 中断系统的移植

Linux 中断系统中面向对象的方法以及两种设计模式的应用降低了 Linux 中断系统与硬件的耦合度, 提高了自身的可移植性. 根据具体的硬件设计, 使用 irq_flow_handler 来包装中断处理代码, 并增加 irq_chip 结构来实现中断硬件的控制代码, 即可在不影响内核其他部分的情况下, 完成中断的移植. 下面以 MP2530F 开发板为例, 介绍中断移植的过程.

2.1　MP2530F 中断系统概况

MP2530F 是韩国美智(MagicEyes)公司专门为嵌入式多媒体处理器 MMSP2+ 设计的多媒体芯片. 它广泛地应用于 GPS 车载导航、移动电视、广告终端等领域. MP2530F 使用 ARM926EJ 和 ARM946E 双 CPU, 一共要处理包括 DMA、UART、I2C、和 GPIO 等 42 个外部中断. MP2530F 有两个独立的中断控制器, 分别处理两个 CPU 的中断信息. 外部中断经过中断仲裁器过滤后, 最高优先级的中断通过中断控制器发送给 CPU. 如图 3 所示.

每个 CPU 的中断由 4 个控制器控制, 它们分别是中断等待控制器、中断优先级控制器、中断屏蔽控制器、中断模式控制器. 每个控制器分别由以LOW、

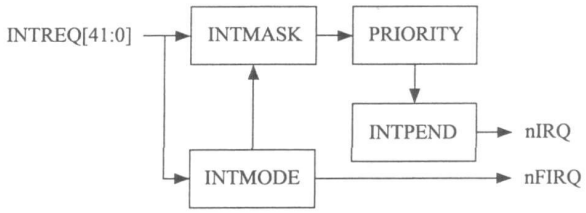

图 3 MP2530F 的中断控制器

Fig. 3 The interrupt controller of MP2530F

HIGH 区分的两个寄存器组成, 标志为 LOW 的寄存器控制 0~ 31 号中断, 标志为 HIGH 的寄存器控制 32 ~ 41 号中断, 剩余的寄存器位保留不用, 这样每个 CPU 能够处理 42 个外部中断源.

2.2 中断解析函数

尽管 MP2530F 可以处理 42 个外部中断源, 仍然无法满足需要. 以 GPIO 为例, MP2530F 拥有 106 个 GPIO 管脚, 每个管脚都可以申请中断, 但 42 个外部中断源中只有一个用于 GPIO 中断, 这些 GPIO 管脚中断全部接在该 GPIO 中断线上. 这样就需要有一个机制对中断进行解析, 分析出产生中断的源.

在硬件上, 中断发生后, 中断等待寄存器上相应的位被置位, 表示该中断被触发. 在内核中, 则需要编写一段中断处理代码来对中断等待寄存器进行分析, 确定实际产生中断的硬件. 在实现上, 可将该中断处理代码包装为一个 irq_flow_handler(图 4 的 mp2530_gpio_irq_demux), 利用它完成 GPIO 中断的解析. 由于内核采用了策略模式, 新增加的 mp2530_gpio_irq_demux 不会对中断系统其他部分造成影响. 该模式的应用如图 4 所示.

使用 mp2530_gpio_irq_demux 后的中断解析过程可以用图 5 来表示.

在图 5 的左侧, 所有的 GPIO 管脚全部接到中断线 IRQ_GPIO 上, 作为 13 号中断发送给 CPU. CPU 收到中断请求后, 内核调用统一接口irq_flow_handler

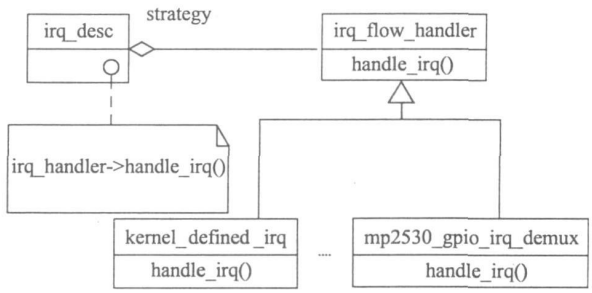

图 4 策略模式的应用

Fig. 4 The application of strategy

进行处理, 在这里就是 mp2530_gpio_irq_demux. mp2530_gpio_irq_demux 将分析各 GPIO 状态寄存器, 寻找中断产生的根源, 计算其中断向量地址, 用 desc_handle_irq 调用其中断处理函数.

使用中断解析函数复用中断线的方法在嵌入式 Linux 中断移植中被广泛应用, 该思想方法也很值得研究和借鉴.

2.3 使用 irq_chip 实现硬件控制

从 Linux2. 6 内核开始, 内核中断系统增加了一个 irq_chip 结构. 该结构在 include\ linux\ irq. h 中定义, 用以在中断初始化与中断处理过程中, 设置屏蔽中断、清除已响应的中断请求等。具体地说, 就是实现中断控制芯片的 mask、unmask 等操作. 通过 irq_chip 结构, 中断硬件操作部分与中断处理被分离开来, 减少了耦合, 同时缩小了 irq_desc[ ]数组的体积.

MP2530F 的中断控制器实际上是由 4 组寄存器组成, 分别控制中断等待、中断屏蔽等功能, 因此它的 irq_chip 实际上就是完成相应寄存器位的设置. 在 irq_chip 抽象类中一共定义了 startup、shutdown 等十几个操作, 一般不需要全部实现, 必须实现的操作有 mask、unmask、ack 等. MP2530F 的中断源包括 DMA、UART、GPIO 等, 以 GPIO 为例, 从抽象类 irq_chip 派生出子类 mp2530_irq_gpio, 对 GPIO 子中断进行操作. mp2530_irq_gpio 实现了以下操作:

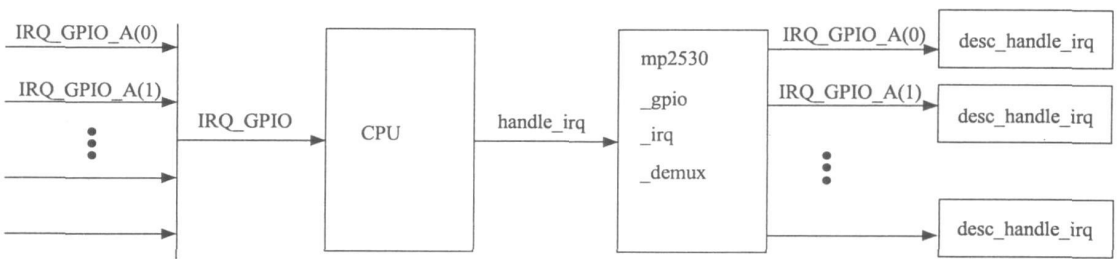

图 5 中断解析过程

Fig. 5 The procedure of irq_demux

• mask: 操作 GPIO 中断屏蔽寄存器, 使 GPIO 屏蔽位置位, 禁止 GPIO 中断.

• unmask: 操作 GPIO 中断屏蔽寄存器, 使 GPIO 屏蔽位清除, 允许 GPIO 中断.

• ack: 设置 GPIO 中断未决(pending) 寄存器, 提示 CPU 处理该中断.

• set_type: 设置中断触发类型, GPIO 中断使用电平触发.

在前面提到, Linux 中断系统中 irq_chip 的实现利用了策略模式的思想. 得益于此, mp2530_irq_gpio 的添加实现不会影响到中断系统的其他部分. irq_chip 的策略模式与 irq_flow _handler 抽象类似, 在此不再赘述.

2.4　Linux 中断移植的验证与应用

Linux 中断系统的移植为嵌入式设备的使用带来了便利, 同时将 CPU 从不必要的等待中解放出来, 提高了系统的效率. Linux 中断在设备驱动程序中有着广泛的应用, 可以利用设备驱动程序来检验和使用移植的中断.

在设备驱动程序中使用中断一般需要完成以下工作: 中断注册、中断服务程序编写和中断释放. 中断的注册与释放在分析中断系统运行时已经介绍过, 中断服务程序的编写则需要遵循简洁、快速的原则, 一般只完成寄存器等信息的读取, 而将其他操作放在软中断或者 tasklet 中完成.

以 MP2530F 的触摸屏驱动为例来说明. 触摸屏按下时产生中断, 驱动程序需要读取按压状态、坐标信息等数据, 并通过 Linux 的 Input-Event 编程接口传递到用户空间[6]. 中断服务程序仅负责读取数据, 而将数据处理放在用户空间来完成, 这样可以减少中断对系统运行效率的影响. 基于以上分析, 触摸屏驱动得以顺利完成, 说明中断移植成功.

3　结　论

本文基于 ARM 处理器, 分析了嵌入式 Linux 的中断系统的结构以及运行过程. 尽管 Linux 中断系统的代码复杂度较高, 但由于在设计中使用了面向对象的方法并使用了策略模式和职责链模式两种设计模式, 从而有效地屏蔽了底层硬件实现的复杂性, 降低了 Linux 中断系统与硬件系统的耦合度, 提高了 Linux 内核的可移植性, 减少了 Linux 中断系统移植的工作量. 在此基础上, 本文以一款基于 ARM9 处理器的嵌入式开发板为例, 说明了在具体的硬件开发板上进行 Linux 中断系统的移植的方法. 尽管嵌入式系统的硬件系统千差万别, 但考虑到 Linux 内核良好的架构设计, 本文对于了解 Linux 内核的中断实现, 以及嵌入式 Linux 的中断移植均有一定的参考价值.

参考文献:

[1] 王莹, 何小庆. 2008 年度嵌入式应用调查报告[J]. 电子产品世界, 2009(1): 11-14.

[2] Robet L. Linux 内核设计与实现[M]. 2 版. 陈莉君, 等译. 北京: 机械工业出版社, 2007.

[3] Linux Kernel Drganization Inc. Linux2. 6. 20 内核源码[EB/OL]. 2007-02-04. http://www.kernel.org.

[4] Erich G, Richard H, Ralph J, et al. 设计模式: 可复用面向对象软件的基础[M]. 李英军, 等译. 北京: 机械工业出版社, 2000.

[5] 毛德操, 胡希明. Linux 内核源码情景分析[M]. 杭州: 浙江大学出版社, 2001.

[6] 郑灵翔. 嵌入式 Linux 系统设计[M]. 北京: 北京航空航天大学出版社, 2008.

Analysis and Porting of ARM Linux Interrupt System

CHEN He-jie, ZHENG Ling-xiang*

(School of Information Science and Technology, Xiamen University, Xiamen 361005, China)

Abstract: Embedded Linux porting is a difficult hotspot in the field of embedded system. This paper analyses the ARM Linux interrupt system, which is very important in embedded Linux porting, by studying its source code. The application of object-oriented and design pattern brings a fine architecture to Linux interrupt system, and makes it easy to port. Taking MP2530F development kit as an example, this paper introduces a method of porting Linux interrupt system and a solution to the problems confronted. It is useful in understanding and porting Linux interrupt system.

Key words: ARM Linux; interrupt; object-oriented; design pattern