嵌入式 Linux 下 CAN 控制器的驱动程序设计

陈祖爵， 周 明
（江苏大学 计算机与通信工程学院，江苏 镇江 212013）

摘 要：嵌入式操作系统 uClinux 下扩展各类 CAN 总线设备，需设计相应的驱动程序。MCP2510 为常用的 CAN 总线控制器，以它为例，详细研究分析了嵌入式操作系统中设备管理和驱动程序的特点，以及 uClinux 下 CAN 设备驱动程序的开发流程和技巧，并结合 CAN 总线技术的特点，设计了相关的重要数据结构和操作代码。最后介绍了把驱动程序编译添加进 uClinux 内核的方法，实现了 CAN 设备的驱动。

关键词：嵌入式系统；CAN 总线；uClinux；设备驱动；MCP2510
中图法分类号：TP368.1 **文献标识码**：A **文章编号**：1000-7024 (2006) 21-4097-04

Driver design of CAN controller in embedded Linux

CHEN Zu-jue, ZHOU Ming
(Institute of Computer and Communication Engineering, Jiangsu University, Zhenjiang 212013, China)

Abstract: To expand all kinds of CAN devices in the embedded operation system of uClinux should design the corresponding driver. MCP2510 is commonly used CAN controller, taking it for example, the management of the device and characteristic of the driver in the embedded operation system are researched and analyzed in detail, including the development procedure and skill of CAN device driver in the uClinux. The relevant important data structure and operating code is written which is combined the technological characteristic of CAN bus. Finally, the method of compiling and adding the driver into uClinux kernel is introduced, the normal work of CAN devices is realized.

Key words: embedded system; CAN bus; uClinux; device driver; MCP2510

0 引 言

CAN 总线技术是现今流行的一种先进的现场总线技术，可以有效的支持分布式控制和实时控制的串行通信网络。由于 CAN 总线具有通信速率高，可靠性高，连接方便和性能价格比高等诸多优点，因此在嵌入式系统开发中有普遍的应用。目前，CAN 总线通信控制芯片众多，要在 uClinux 平台下开发基于 CAN 总线的应用系统，就需要自己开发 uClinux 下的驱动程序。本文将基于一个 CAN 总线在汽车电子中的应用详细介绍在 uClinux 下 CAN 总线控制器驱动程序的设计过程。

1 系统硬件结构

本嵌入式系统主要的硬件组成为：处理器采用三星公司的 S3C44B0X，CAN 总线控制器和收发器分别采用 MicroChip 公司的 MCP2510 和 MCP2551。开发一个 uClinux 的驱动，在熟悉 uClinux 内核结构之外，大量的工作在于阅读相应的控制芯片手册。硬件信息决定驱动的主要结构。S3C44B0X 采用的是 ARM 公司的 16/32 位 ARM7TDMI 内核，它是三星公司为一般应用提供的高性价比和高性能的微控制器解决方案，特别适合对成本和功耗敏感的应用场合。MCP2510 是一款带有符合工业标准的 SPI 接口的 CAN 总线控制芯片，它支持 CAN 技术规范 V2.0A/B，并能够发送和接收标准的和扩展的信息帧，同时具有接收滤波和信息管理的功能。MCP2510 在目前市场上是体积最小、最易于使用也是最节约成本的独立 CAN 控制器。MCP2551 是与 MCP2510 相配的高速 CAN 总线收发器，它担负着节点和总线之间接收和发送电平转换的任务。

MCP2510 通过 SPI 接口与 S3C44B0X 进行数据传输，最高数据传输数率可达 5 Mb/s。MCP2510 再通过 CAN 收发器连接到 CAN 总线上，CAN 总线上可以挂接多个节点，S3C44B0X 通过 MCP2510 与 CAN 总线上的其它微处理器进行通信。MCP2510 内含 3 个发送缓冲区和两个接收缓冲区，同时具有灵活的中断管理能力，帧屏蔽和过滤、帧优先级设定等特性，这使得微处理器对 CAN 总线的操作变得非常简便。系统原理如图 1 所示。

2 CAN 总线应用系统的软件设计

2.1 嵌入式操作系统选择 uClinux

uClinux 是 Linux2.0 版本得一个分支，被设计用在微型控

收稿日期：2005-09-29。
作者简介：陈祖爵（1953－），男，上海人，副教授，硕士生导师，研究方向为网络技术与嵌入式系统开发应用；周明（1982－），男，上海人，硕士研究生，研究方向为嵌入式系统与计算机网络控制技术。

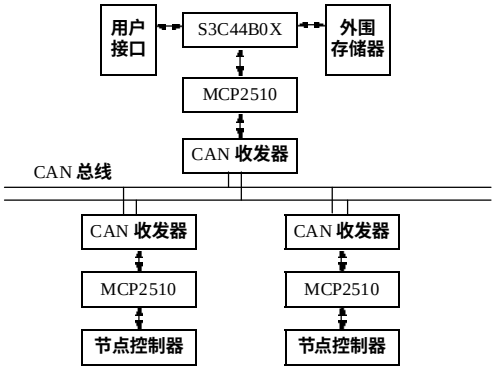

图 1　嵌入式应用系统结构

制应用领域。uClinux 具备标准 Linux 系统的稳定性，并且支持 Linux 内核约定的全部特性。uClinux 同标准 Linux 的最大的区别就是在于内存管理。标准 Linux 是针对有内存管理单元(memory management unit，MMU)的处理器设计的。在这种处理器上，虚拟地址被送到 MMU，MMU 把虚拟地址映射为物理地址。嵌入式应用对成本和实时性敏感，其使用的 CPU 中有很多都没有 MMU，例如本系统采用的 S3C44B0X 就是一款不带 MMU 的微处理器。标准 Linux 无法适用于这部分嵌入式应用。uClinux 通过对标准 Linux 中内存管理的改写，去掉了对 MMU 的依赖，保存了 Linux 内核的大多数优点，因此它在嵌入式应用中有很好的前景。uClinux 的应用主要体现在驱动程序的编写和上层应用软件的编写。所以，针对 CAN 总线控制器 MCP2510 的驱动程序需要我们自己编写。

2.2　CAN 总线控制器驱动程序编写

驱动程序是应用程序与硬件之间的一个中间软件层。它使某个特定的硬件响应一个定义良好的内部编程接口，同时完全隐蔽了设备的工作细节。用户通过一组标准化的调用来完成相关操作，这些标准化的调用是和具体设备驱动无关的，而驱动程序的任务就是把这些调用映射到具体设备对于实际硬件的特定操作上。驱动程序应该为应用程序展现硬件的所有功能，不应该强加其它的约束，对于硬件使用的权限和限制应该由应用程序层控制。驱动程序设计主要需要考虑下面 3 个方面：提供尽量多的选项给用户；提高驱动程序的速度和效率；尽量使驱动程序简单，使之易于维护。

uClinux 支持的设备驱动可分为 3 种：字符设备，块设备，网络接口设备。MCP2510 就属于字符设备。字符设备是 uClinux 中最简单的设备，所谓字符设备就是以字节为单位逐个进行 I/O 操作的设备。在 uClinux 中它们被映射为文件系统的一个节点，这个设备就像是一个普通文件，应用程序使用标准系统调用对它进行打开(open)、读取(read)、写入(write)和关闭(release)等操作。

2.2.1　驱动程序中定义的主要数据结构

驱动程序中读写函数需要传输多个 CAN 消息，我们根据 CAN 通信协议和系统应用的需要，设计一个称为 CanData 的结构体来定义所传输的数据

```
struct {
    unsigned int id;
    unsigned char data [8];
    unsigned char dlc;
    int IsExt;
    int rxRTR;
}CanData;
```

其中 id 为 CAN 消息 ID 号；data 是要传输的消息数据，最大是 8 个字节；dlc 表示实际传输的数据长度，取值范围为 0 到 8；IsExt 是判断 CAN 消息是否使用扩展 ID；rxRTR 是判断该消息是数据帧还是远程帧。

MCP2510 中有 3 个发送缓冲区，可以循环使用，也可以只使用一个发送缓冲区，但是必须保证在发送的时候，前一次的数据已经发送结束。两个接收缓冲区也是一样。处理器通过 SPI 接口对缓存区进行读取和写入。MCP2510 对 CAN 总线的数据发送没有限制，只要用处理器通过 SPI 接口将待发送的数据写入 MCP2510 的发送缓存区，然后再调用 RTS(发送请求)命令即可将数据发送到 CAN 总线上。而对 CAN 总线上的数据接收是通过两个接收缓冲区，两个接收屏蔽器，6 个接收过滤器的组合来实现的。CAN 总线上的帧只有同时满足至少任意一个接收屏蔽器和一个接收过滤器的条件才可以进入接收缓冲区。这里定义了一个 MCP2510_REV 的数据结构，用于记录接收缓冲区运行的各种状态

```
struct {
    CanData MCP2510_Candata [128];
    int nCanRevpos;
    int nCanReadpos;
    int loopbackmode;
    wait_queue_head_t wq;
    spinlock_t lock;
} MCP2510_REV;
```

该结构中首先定义了一个接收缓冲区，nCanRevpos 和 nCanReadpos 分别表示接收缓冲区和用户读取缓冲区数据的状态；loopbackmode 表示支持回环模式，该模式可使器件内部发送缓冲器和接受缓冲器之间进行报文自发自收；wq 定义的是一个等待队列，包含一个锁变量和一个正在睡眠进程链表，作用是当有好几个进程都在等待某件事时，uClinux 会把这些进程记录到这个等待队列；lock 定义的是自旋锁，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。而其它需要锁的函数则会循环查询锁是否可用，作用是实现互斥访问。

2.2.2　驱动程序的接口

驱动程序的接口主要分为 3 部分：①与设备的接口，完成对设备的读写等操作；②与内核通信的接口，由 file_operations 数据结构完成；③与系统启动代码的接口，完成对设备的初始化。

uClinux 继承了 Linux 操作系统下用户对设备的访问方式。设备驱动与内核通信时使用的是设备类型，主次设备号等参数，但是对于应用程序的用户来说比较难于理解和记忆，所以 uClinux 使用了设备文件的概念。它抽象了对硬件文件的管理，为用户程序提供了一个统一的、抽象的虚拟文件系统(virtual file system，VFS)界面。如图 2 所示，VFS 主要由一组标准的，抽象的文件操作构成，以系统调用的形式提供给用户，如 open()、release()、read()、write()、ioctl()等。

图 2　文件层次结构

内核是通过主设备号这个变量将设备驱动程序和设备文件相连的，而构成驱动程序的一个重要数据结构就是 file_operations，内核就是通过这个结构来访问驱动程序的。file_operations 结构定义于 linux/fs.h 文件中，它包含指向驱动程序内部大多数函数指针，它的每一个成员名称对应着一个系统调用。系统引导时，内核调用每一个驱动程序的初始化函数，将驱动程序的主设备号以及程序内部的函数地址结构的指针传输给内核。这样，内核就能通过设备驱动程序的主设备号索引访问驱动程序内部的子程序，完成打开，读写等操作。下面给出了 file_operations 结构体中的一些主要成员

```
struct file_operations {
struct module *owner;// module 的拥有者
loff_t (*llseek) (struct file *, loff_t, int);//移动文件指针的位置，只能用于可以随机存取的设备
ssize_t (*write) (struct file *, const char *, size_t, loff_t *);//向字符设备中写入数据
ssize_t (*read) (struct file *, char *, size_t, loff_t *);//从设备中读取数据，与 write 类似
unsigned int (*poll) (struct file *, struct poll_table_struct *);//用于查询设备是否可读写或处于某种状态
int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);//控制设备，除读写操作外的其它控制命令
int (*mmap) (struct file *, struct vm_area_struct *);//用于把设备的内容映射到地址空间
int (*lock) (struct file *, int, struct file_lock *);//文件锁定，用于文件共享时的互斥访问
int (*open) (struct inode *,  struct file *);  //打开设备进行 I/O 操作
int (*release) (struct inode *,  struct file *);  //关闭设备并释放资源
…
};//file_operations 结构中的成员全部是函数指针，所以实质上就是函数跳转表
```

由于在 file_operations 结构中，每个字段都必须指向驱动程序中实现特定操作的函数，可以想象，随着内核新功能不断的增加，file_operations 结构也就会变得越开越庞大。所以，现在通常采用“标记化”的方法来为该结构初始化，即对驱动中用到的函数记录到相应字段中，没用到的就不管，这样代码就精简了许多。代码片断如下

```
static struct file_operations s3c44b0x_fops = {
    owner:     THIS_MODULE,
    write:          mcp2510_write,
    read:           mcp2510_read,
    ioctl:          mcp2510_ioctl,
    open:           mcp2510_open,
    release:   mcp2510_release,
};
```

要注意的是这种表示方法不是标准 C 语法，而是 uClinux 下 GNU 编译器的一种特殊扩展。它使用函数名对各结构字段初始化，好处是结构清晰，易于理解，并且避免了结构发生变化带来的许多问题。上面代码中，owner 声明模块的拥有者，mcp2510_write 和 mcp_2510_read 负责对缓冲区读写数据，mcp2510_open 负责打开 CAN 总线控制器，并清空 3 个发送缓冲区，mcp2510_release 负责关闭 CAN 总线控制器，mcp2510_ioctl 负责向 CAN 总线控制器发送各种控制命令。mcp2510_write()代码片段编写如下

```
static ssize_t mcp2510_write(struct file *file, const char *buffer, size_t count, loff_t *ppos)
//*filp 为打开的文件，*buffer 为数据缓存，count 为请求传送数据长度，*ppos 为用户在文件中进行存储操作的位置
{char sendbuffer[sizeof(CanData)];
    if(count==sizeof(CanData)){//根据发送数据的长度，以两种模式发送数据
        copy_from_user(sendbuffer, buffer, sizeof(CanData));//将数据从应用数据空间拷贝到内核
        MCP2510_canWrite((PCanData)sendbuffer);
        …
    }
    if(count>8)
    return 0;
    copy_from_user(sendbuffer, buffer, count);
    MCP2510_canWriteData(sendbuffer, count);
  …
}
```

2.2.3　驱动程序的初始化与设备注册

定义并初始化完成 file_operations 结构后，下面必须定义一个初始化函数，这里我们定义了一个名为 mcp2510_init()的函数。在 uClinux 初始化的时候要调用该初始化函数。初始化函数要完成的任务很多，主要有以下几点：

(1)初始化设备相关的参数。对 MCP2510 来说，这里主要完成 CAN 总线波特率的设置，ID 过滤器的设置，清空接收和发送缓冲区，开启中断等工作。

(2)注册设备。注册设备所使用的函数原型是

```
int register_chrdev (unsigned int major,  const char *name, struct file_operations *fops)
```

其中 major 是主设备号，name 是设备名称，fops 就是内核访问设备的接口。前面提到 uClinux 内核是通过主设备号将设备驱动程序和设备文件相连。uClinux 支持的主设备号有限，2.0 以前版本的内核支持 128 个主设备号，在 2.4 版内核中已经增加到 256 个。为了不造成使用上的混乱，对主设备号的分配

常常采用动态分配的方法，即在用 register_chrdev () 注册模块时，给 major 参数赋值为 0，这样系统就会在所有未被使用的设备号中为我们选定一个，作为函数的返回值返回给我们。注册设备代码片断编写如下

```
#define DEVICE_NAME    "s3c44b0x-mcp2510"
…
ret = register_chrdev(0, DEVICE_NAME, &s3c44b0x_fops);
```

(3)注册设备使用的中断。因为中断信号往往是通过特定的中断信号线传输的，任何一款芯片留给中断信号的接口都是有限的，所以内核会维护一个中断信号线注册表，模块要使用中断就得向它申请一个中断通道，当它使用完该通道之后要释放该通道。这里使用的就是函数

request_irq(IRQ_MCP2510, s3c44b0x_isr_mcp2510, SA_INTERRUPT, DEVICE_NAME, isr_mcp2510);

其中 IRQ_MCP2510 是请求的中断号；s3c44b0x_isr_mcp2510 是中断处理函数的指针；SA_INTERRUPT 是一个与中断管理有关的位掩码选项；DEVICE_NAME 是设备名，它被用来在/proc/interrupts 中显示中断拥有者；isr_mcp2510 是一个惟一的标志符，通过该指针多个设备可共享信号线，驱动程序也可用它指向自己的私有数据区，用来识别哪个设备产生了中断。

3 设备驱动程序的编译和添加

在 uClinux 下，对驱动程序的编译添加一般有两种方式。可以静态编译进内核，再运行新的内核来测试；也可以编译成模块在运行时加载。第 1 种方法效率较低，但在某些场合是惟一的方法。模块方式调试效率很高，它使用 insmod 工具将编译的模块直接插入内核，如果出现故障，可以使用 rmmod 从内核中卸载模块。不需要重新启动内核，这使驱动调试效率大大提高。但嵌入式系统是针对具体应用的，所以一般采用将设备驱动程序以静态的方法编译进内核。具体步骤如下：

(1)将驱动 mcp2510.c 添加到/uclinux_dist/linux/drivers/char 目录之下；

(2)修改该目录下的 mem.c 文件，在 int chr_dev_init()函数中增加如下代码：

```
#ifdef CONFIG_S3C44B0X_MCP2510
mcp2510_init ();
#endif
```

(3)修改该目录下的 Makefile 文件，增加如下代码：

```
ifeq ($(CONFIG_S3C44B0X_MCP2510),y)
L_OBJS+=mcp2510.o
endif
```

(4)修改/uclinux_dist/linux/arch/armnommu 目录下 config.in 文件，在 comment'Character devices'语句下面加上：

bool'Add CAN Controller MCP2510'CONFIG_S3C44B0X_MCP2510

(5)编译内核。我们在配置字符设备时就会有选项 Add CAN Controller MCP2510，当选中这个选项的时候，设备驱动就加到内核中去了。编译成功后，就可以像使用普通文件一样对设备进行操作，在编写的应用程序中先打开设备再读写数据。

4 结束语

本文详细介绍了在嵌入式操作系统 uClinux 下 CAN 总线控制器 MCP2510 驱动程序的设计开发过程。该驱动程序根据 CAN 总线的通讯协议和总线控制器 MCP2510 的工作特点，合理的设计了数据结构和缓存区控制方法，并结合 uClinux 下驱动程序编写的一般规则，编写了相关的操作代码，实践证明该驱动程序正确可行。CAN 总线是一种广泛应用的优秀现场总线技术，再借助源码开放的 uClinux 在嵌入式开发中的特点与优势，开发工作者就可以灵活快捷的开发各类相关产品。

参考文献：

[1] MicroChip. MCP2510 stand - alone CAN controller with SPI interface [M]. *Microchip*, 2002.

[2] 邬宽明. 现场总线技术应用选编[C]. 北京:北京航空航天大学出版社, 2003.

[3] 邹思轶. 嵌入式 Linux 设计与应用[M]. 北京:清华大学出版社, 2002.

[4] 王学龙. 嵌入式 Linux 系统设计与应用[M]. 北京:清华大学出版社, 2002.

[5] 杨波,徐成,李仁发. 嵌入式 LINUX 上的 CAN 设备驱动程序的设计[J]. 科学技术与工程, 2004,4(12):1019-1022.

[6] Alessandro Rubini, Jonathan Corbet.LINUX 设备驱动程序[M]. 北京:中国电力出版社, 2002.

[7] 龚彬,吴平,刘维亚,等. 基于 uClinux 嵌入式系统的设备驱动程序的研究[J]. 电子工程师, 2004,30(1):67-70.

[8] 陆宝辂,邵贝贝, 李荐民. uClinux 的设备驱动程序开发[J]. 单片机与嵌入式系统应用, 2003,(6):81-83.

[9] 苗启广,魏乐,王宝树,等. 基于 uClinux 的嵌入式软件开发架构[J]. 计算机工程与设计, 2004,25(6):881-883.

[10] 柳谦,刘震宇,龙剑飞. 一种可靠的 CAN 总线多点通信设计方法[J]. 计算机工程与设计, 2005,26(5):1323-1326.

(上接第 4096 页)

[5] Edward Kit.软件测试：过程改进 [M]. 第 2 版. 北京:机械工业出版社, 2003.

[6] Gerard O'Regan.软件质量使用方法论[M].北京:清华大学出版社, 2004.

[7] Filippo Lanubile. A collaborative tool for geographically dispersed inspection teams [C]. *Cologne, Germany: Proceeding of 4th ICSTEST International Conference on Software Testing*, 2003.

[8] Philip M Johnson, Danu Tjahjono. Improving software quality through computer supported collaborative review [C]. *Milan, Italy: Proceedings of the Third European Conference on Computer Supported Cooperative Work*, 1993.