基于 ARM 的 Linux 下 LonWorks 总线设备驱动设计

梁惺彦，顾 晖
（南通大学 计算机科学与技术学院，江苏 南通 226019）

摘 要：利用神经元芯片 CYC53120 和 S3C2410 芯片，实现嵌入式平台下 LonWorks 总线的运用；同时介绍嵌入式 Linux 下设备驱动程序的构成，描述了 LonWorks 设备驱动程序的软件框架，为嵌入式 Linux 设备的开发提供借鉴。

关键词：嵌入式 Linux；LonWorks；设备驱动
中图分类号：TP336 文献标识码：B

The design of LonWorks bus device-driver based on ARM Linux

LIANG Xing Yan，GU Hui
(College of Compurer Science and Technology，Nantong University，Nantong 226019，China)

Abstract：This paper introduces how to use the neuron chip CYC53120 and the ARM9 chip S3C2410 to realizes the LonWorks bus based on the embedded platform，The it introduces the device-driver constitution of embedded Linux，and describes the software frame of the LonWorks device-driver.It provides a model for the device development of the embedded Linux.

Key words：embedded Linux；LonWorks；device-driver

LonWorks 是美国 Echelon 公司 1992 年推出的局部操作网络，最初主要用于楼宇自动化，但很快发展到工业现场网。LonWorks 技术为设计和实现可互操作的控制网络提供了一套完整、开放、成品化的解决途径，它协议完整、通信可靠，而且为用户提供了功能强大的开发工具(LONBU ILDER，NODEBU ILDER)。

在 LonWorks 现场总线设备的使用过程中，由于其设备驱动与操作系统的相关性，从而要求开发者在开发过程不仅实现硬件构成，更需要熟悉操作系统及设备驱动程序的制定。本文给出在 ARM 平台下实现 LonWorks 总线设备的互联，并在嵌入式 Linux 系统下，介绍 LonWorks 现场总线设备驱动程序的设计与实现。

1 LonWorks 总线设备的构成

LonWorks 技术的核心是神经元芯片(Neuron Chip)。该芯片内部装有 3 个微处理器：MAC 处理器完成介质访问控制；网络处理器完成 OSI 的 3～6 层网络协议；应用处理器完成用户现场控制应用。它们之间通过公用存储器传递数据。同时神经元芯片共设置 11 个 I/O 口，这些 I/O 口可根据不同需求，利用 Neuron C 编程来灵活配置与外围设备的接口，如 RS232、并口、定时/计数、位 I/O 等。其芯片结构如图 1 所示[1]。

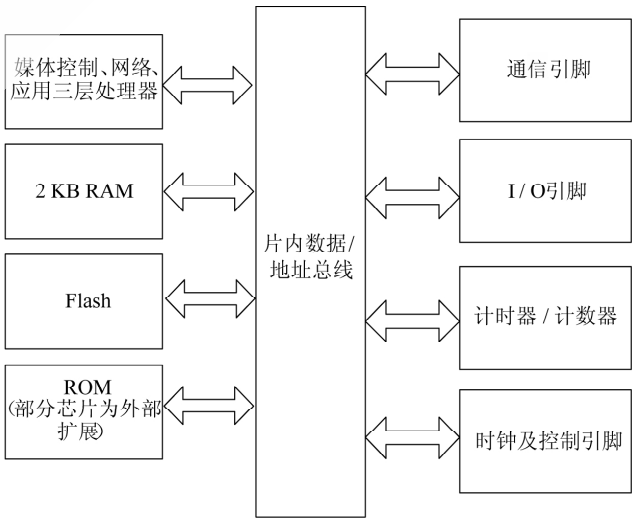

图 1 Neuron 芯片 CY7C53120 基本结构

在本系统设计中，基于嵌入式 ARM 平台下实现对 LonWorks 总线的访问，设计原理是利用 S3C2410 芯片的

SPI(Serial Peripheral Interface)接口[2]与 Neuron 芯片来实现数据通信，其原理如图 2 所示。

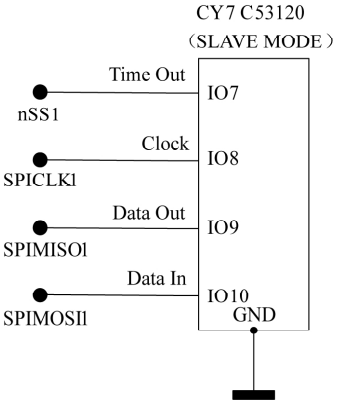

图 2 S3C2410 与 CYC53120 的 SPI 接口图

SPI 总线系统可直接与各个厂家生产的多种标准外围器件直接接口，它有 4 条引脚：SSEL(从器件选择线)、MOSI(主机输出、从机输入数据线)、MISO(主机输入、从机输出数据线)、SCK(同步串行时钟线)。S3C2410 包含 2 个 SPI 接口，本文利用 SPI1 与 Neuron 芯片 CY7C53120 相连。

2 Linux 下设备驱动程序

设备驱动程序是操作系统内核与机器硬件之间的接口。在 Linux 中，设备驱动程序为应用程序屏蔽了硬件的细节，对应用程序而言，硬件设备只是一个设备文件，可以通过相应的系统调用像操作普通文件一样对硬件设备进行操作。

Linux 系统的设备分为字符设备(char device)、块设备(block device)和网络设备(network device)3 种[3-4]。字符设备是指存取时没有缓存的设备；块设备的读写则都有缓存来支持，只能以块为单位进行读写，并且块设备必须能够随机存取(random access)；而 Linux 的网络设备开发则主要基于 BSD Unix 的 socket 机制。本文要开发的 Lonworks 设备驱动程序是一个字符型的设备，其基本组成如图 3 所示。

Linux 设备驱动程序可以分为设备初始化子程序及卸载程序、服务子 I/O 请求的子程序和中断服务子程序 3 个主要组成部分：

(1)设备初始化子程序及卸载程序。Init_module 用以负责检测所要驱动的硬件设备是否存在和是否能正常工作。如果该设备正常，则对这个设备及其相关的设备驱动程序需要的软硬件进行初始化，其初始化程序流程如图 4 所示。

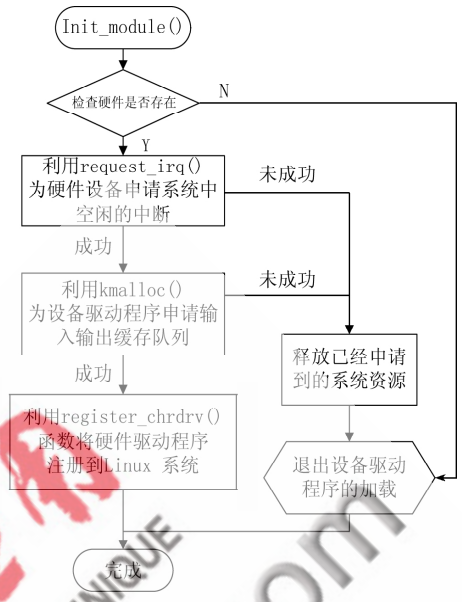

图 4 设备初始化程序流程

Cleanup_module 用以完成卸载设备时要做的工作，其设备卸载流程如图 5 所示。

图 5 设备卸载流程

(2)服务于 I/O 请求的子程序，又称为驱动程序的上半部。应用程序可以通过系统来调用此部分程序。此部分程序在执行时，系统仍认为是与应用程序进程属于同一个进程，具有进行此系统调用的用户程序的运行环境，只是由用户态变成了核心态，因而可以在其中调用 sleep()等与进程运行环境有关的函数。

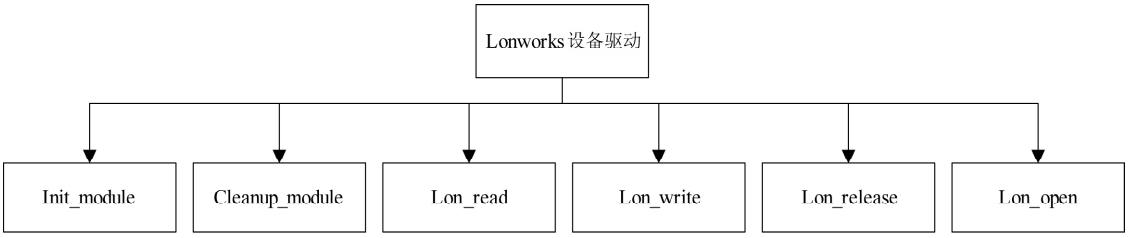

图 3 Lonwork 设备驱动软件基本框架

此部分的设计主要是对 file_operations 结构的各个入口点的实现。从而实现了支持文件系统的调用(如 open，close，read 等)。file_operations 结构是 Linux 操作系统中用于实现驱动程序的最重要的数据结构，对 Linux 提供 I/O 请求的子程序的一系列入口点进行了封装。下面给出用于 Lonworks 设备驱动的 file_operations 结构示例。

```
struct file_operations{
int(*lseek)(struct inode *inode,struct file *flip,off_t off,int pos);
int(*read)(struct inode *inode,struct file *filp,char *buf,int count);
int(*write)(struct inode *inode,struct file *filp,char *buf,int count);
int(*ioctl)(struct inode *inode,struct file *filp,unsigned int cmd,unsigned int arg);
int(*open)(struct inode *inode,struct file *filp);
viod(*release)(struct inode *inode,struct file *filp);
}
```

由上可见 file_operations 结构中的成员全部是函数指针，该结构实质上就是函数跳转表。每个应用进程对设备的操作，都可以根据设备号，转换成对 file_operations 结构的访问，通过调用相关函数完成具体操作。

(3)中断服务子程序，又称为驱动程序的下半部。在 Linux 系统中，并不直接从中断向量表中调用设备驱动程序的中断服务子程序，而是由 Linux 系统来接收硬件中断，再由系统来调用中断服务子程序。

中断可以在任何一个进程运行时产生，因而中断服务程序的调用，不依赖于任何进程的状态，即不能调用任何与进程运行环境有关的函数。在系统内部，I/O 设备的存/取通过一组固定的入口点来进行，这组入口点是由每个设备的驱动程序提供的。对于本文涉及的 LonWorks 设备驱动，下面分析其作为字符型设备驱动程序所需用的常用入口点。

Lon_open(struct inode *inode，struct file *filp)入口点用以打开设备准备 I/O 操作。对字符设备文件进行打开操作，都会调用设备的 open 入口点。open 子程序必须对将要进行的 I/O 操作做好必要的准备工作，如清除缓冲区等。Lon_release(struct inode *inode，struct file *filp)入口点用以关闭设备。当最后一次使用设备终结后，调用 release 子程序。

Lon_read(struct inode *inode，struct file *filp，char *buf，int count)入口点用以从设备上读数据。可以利用所设定的缓冲区进行 I/O 操作，从缓冲区里读数据。对设备进行读操作将调用 read 子程序。相似的，Lon_write(struct inode *inode，struct file *filp，char *buf，int count)入口点用以往设备上写数据。对设备进行写操作将调用 write 子程序。

Lon_ioctl(struct inode *inode，struct file *filp，unsigned int cmd，unsigned int arg)入口点执行读、写之外的一些硬件控制操作，操作命令代码通过 cmd 参数传送，命令参数通过 arg 参数传送。此函数也为功能扩展提供接口。

实现了以上 3 部分，核心驱动模块部分就完成了。对以上模块进行编译并加载后，Linux 用户可用 mknod 命令利用动态分配的主设备号建立相应的设备文件，并对它设置恰当读写权限后，就可以在应用程序中通过这个设备文件来操作 LonWorks 总线设备了。这样做不仅使得应用程序编程风格更加统一，代码更具鲁棒性，应用系统更加安全更易于维护，而且可在核心级来保证关键部分的实时响应，从而降低用户程序开发的难度。

以上介绍了在 ARM9 平台的嵌入式 Linux2.6.x 系统中实现 Lonwork 通信的一种方法，并在以武汉创维特公司的 JXARM9-2410 实验箱为基础扩展的硬件平台上实验完成。后期可在此基础上，结合 Linux 在网络通信中的特长，实现对 Lonwork 网络的应用管理或与其他通信网络的互连，有较好的实用性。

参考文献

[1] Cypress Semiconductor Corporation.Neuron chip technical reference manual[M/OL].Http://www.cypress.com.2003.

[2] SAMSUNG公司.S3C2410X RISC Microprocessor Reference Manual.2003.

[3] 杨峰，吉吟东，薛明.CAN通信卡的Linux设备驱动程序设计实现[J].电子技术应用，2002，28(1)：52-55.

[4] 于明，范书瑞.ARM9嵌入式系统设计与开发教程[M].北京：电子工业出版社，2006.

(收稿日期：2009-02-17)