基于 QNX 的 PC104 总线设备驱动模块的开发

Development of Drivers for PC104 Devices Based on QNX OS

姜广山　祖家奎　（南京航空航天大学自动化学院，江苏 南京 210016）

摘　要

QNX 作为最优秀的嵌入式实时操作系统之一，已被应用于诸如医疗仪器、电网通信以及航空航天等任务关键型领域。在掌握 QNX 微内核结构和消息传递机制的基础上，论述了 QNX 下资源管理器的原理以及资源管理器与设备驱动程序的关系，具体分析了 PC104 总线设备驱动程序编写规范和步骤。实现了 QNX 下应用程序对硬件访问的无关性，大大增强了系统安全性以及移植性等。

关键词：QNX，PC104 总线，消息传递，资源管理器，设备驱动

Abstract

On the basis of microkernel architecture and message passing,this paper introduces the relation between resource managers and device drivers and analyses a general method of programming PC104 device drivers.It achieves the independence between applications and device drivers.Meanwhile,it enhances the security and portability of QNX.

Keywords:QNX,PC104,message passing,resource manager,device driver

QNX 是由加拿大 QNX 软件系统公司开发的实时操作系统软件，支持多任务和多用户的嵌入式实时操作系统，其建立在真正微内核结构和完全地址保护的基础之上。

为了充分发挥 QNX 实时操作系统良好的裁剪型以及模块化程度高等优点，本文从 QNX 微内核体系结构和消息传递机制出发，介绍了基于资源管理器机制的 PC104 总线设备驱动的开发流程和实现方法。通过符合 POSIX 标准的驱动程序开发，实现了用户程序对硬件设备访问的平台无关性和可移植性，便于应用程序开发的模块化和规范化，并保证系统软件的安全性和实时性。

1　QNX 实时操作系统与设备驱动

1.1 QNX 微内核结构

QNX 是真正的微内核系统。图 1 是 QNX 体系结构图，整个系统由一个微内核和基于内核管理的多个系统进程模块组成。实现这种结构的关键是消息传递机制。QNX Neutrino 各模块之间是通过消息传递来进行通信，而内核只提供四种服务：进程调度、进程间通信、底层网络通信和中断处理，并通过消息传递与系统的各模块进行通信。

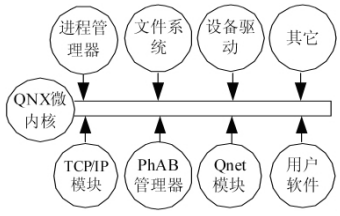

图 1　QNX 微内核体系结构示意图

QNX 微内核体系结构和基于消息的进程间通信机制，使内核之外的各个模块可以任意的删除和添加，且模块之间互不影响，保证了其具有良好的裁剪性、移植性和较高模块化程度。

1.2 消息传递机制

消息传递是一种阻塞式通信，发生在两个进程之间，以客户/服务器模式实现。两个进程分别为客户进程和服务器进程，由客户进程向服务器请求服务。

QNX Neutrino 中，消息传递的体系结构可分为三个过程：①客户进程向服务器发送消息；②服务器接收客户进程消息并根据消息进行相应处理；③服务器回应客户进程。消息传递是通过 Neutrino 库函数完成的，客户只需要简单的调用 POSIX/ANSIC 函数就可以完成消息的传递。

1.3 QNX 设备驱动开发模式

QNX 下硬件设备驱动的开发可以分为两种模式。一种是在应用程序中直接对硬件 I/O 口进行操作；一种是利用消息传递和资源管理器机制，应用程序通过文件接口来实现对硬件的操作。

1)在应用程序中调用硬件初始化函数并直接对硬件 I/O 端口进行读写和控制。此开发模式易于实现，但由于其对硬件地址操作存在于应用程序中，不利于应用程序在不同平台的移植；

2)利用消息传递和资源管理器机制实现设备驱动开发。此模式下，驱动程序作为服务器进程，可以独立于内核编译，保证系统的安全性。另外，调试过程简单。同时为用户程序提供了标准设备接口，简化了应用程序的编写，增加了程序的可读性和移植性。本文内容重点讲述此种开发模式。

2　QNX 设备驱动体系结构

利用消息传递和资源管理器机制开发的硬件驱动程序向应用程序屏蔽了硬件在实现上的具体细节，使得应用程序可以像操作普通文件一样来操作外部设备。也就是说，QNX 抽象了对硬件的处理，使客户程序可以通过标准的系统调用接口函数来完成对设备的打开、关闭以及读写等操作。

2.1 设备驱动与资源管理器的关系

QNX 下所有的设备都称为资源，而所有资源都由资源管理器进行管理，所以设备驱动程序又被称为设备资源管理器。设备资源管理器负责为不同类型的设备提供一个接口。可以说，在 QNX 中，设备驱动程序与资源管理器没有本质的区别。

2.2 设备资源管理器工作机制

设备资源管理器实际上是一个用户级的服务器进程，它接收来自客户程序发送的消息，并根据消息类型有选择性的与硬件进行通信。为了更加直观，这里通过资源管理器的伪代码来阐述其结构体系与工作机制。

图 2 是资源管理器的伪代码结构，资源管理器体系结构包含三部分内容：①初始化资源管理器，最主要的工作是建立一条通道，使客户程序能够向资源管理器发送消息；②资源管理器向进程管理器注册路径名；③循环接收消息并进行处理。

资源管理器能够接收两类消息：连接(connect)消息和 I/O 消息。连接消息是客户程序发出的、执行基于路径名操作的消息

(如 _IO_OPEN 消息);而 I/O 消息是依赖于客户进程与资源管理器之间的上下文并进行基于 I/O 消息的处理(如 _IO_READ 消息)。QNX 提供了相应的默认消息处理函数,用户也可以在资源管理器中进行自定义。

```
Initialize the resourse manager                    (1)
Register the name with the progress manager (2)
Do forever
    receive a message
    SWITCH on the type of message                  (3)
        CASE _IO_CONNECT:
            call io_open handler
            ENDCASE
        CASE _IO_READ:
            call io_read handler
            ENDCASE
        CASE _IO_WRITE:
            call io_write handler
            ENDCASE
    ENDSWITC
ENDDO
```

图 2 资源管理器伪代码结构图

资源管理器与客户程序的连接是通过路径名的空间映射来实现的。当客户程序调用标准 POSIX/Neutrino 库函数通过资源管理器建立的路径空间对设备进行操作时,库函数一般会构建事先定义好的消息并将此消息发送给相应的资源管理器,资源管理器调用相应消息处理函数对不同类型的消息进行处理。在设备资源管理器中,消息处理过程就是对硬件设备的操作过程。

2.3 设备驱动程序体系结构

一般来讲,设备驱动程序包括两层:①硬件/抽象屏蔽层,简称 HSL;②硬件获取层,简称 HAL。HSL 层提供了硬件的逻辑接口,HAL 层实现对硬件操作。

QNX 下设备驱动程序体系结构参考图 3。可以看出,QNX 采用资源管理器机制清楚的划分了 HAL 和 HSL 这两层结构。其中,用户程序在 HSL 层可以通过 POSIX 标准函数对设备以文件形式进行访问,这样就可以做到用户程序对设备访问的无关性和可移植性;而 HAL 层,也就是设备的资源管理器,它按照 HSL 层发送的消息对设备的硬件进行操作,并将结果返回 HSL 层。

3 PC104 总线设备驱动模块的实现

PC104 总线设备驱动模块的实现主要相关设备资源管理器的编写,它主要实现以下几个功能:①建立与客户进程的交互;②设备初始化;③消息处理函数的编写;④数据与状态的返回。图 3 是 QNX 下 PC104 设备驱动实现流程示意图。

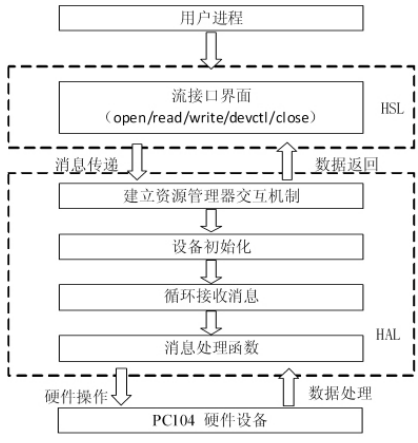

图 3 QNX 下 PC104 设备驱动实现流程

3.1 建立与客户进程的交互

资源管理器本质上是一个用户级服务器,它负责接收并处理客户进程发送的消息。在这个过程中,资源管理器通过建立用来接收消息的通道、设备文件名的注册以及消息函数的初始化等工作来建立与客户进程的交互。具体实现方法为:

1)资源管理器通过初始化 dispatch 接口(通过调用 dispatch_create()函数实现)来建立与客户进行消息传递的机制。

2)资源管理器能够接收消息之前,还需要进行设备文件路径名的注册(通过函数 resmgr_attach()来实现)。注册完之后,其它的进程均可以通过此路径与设备文件相关联。

3)资源管理器作为服务器一直处于 Receive 阻塞状态(通过函数 dispatch_block()启动消息接收循环),直到当客户对设备文件进行操作,将消息发送到资源管理器时,资源管理器进入就绪态。资源管理器根据接收到客户进程发送的消息,对消息进行解码(调用 dispatch_handler()函数),并调用相应的处理函数。

3.2 设备初始化

设备的初始化是指板卡设备基地址的识别以及硬件资源工作模式设定等。对于 PCI 设备和属于 PNP 类型的 ISA 设备,QNX 提供了相应的函数调用,可以确定设备在系统中的资源分配。而 PC104 总线规范的设备,其配置由板卡所决定。具体初始化过程是在资源管理器初始化中完成的,可以分为下面三个部分:

1)设置板卡基地址。基地址通常由板卡跳线决定。设置时需要注意不能与 CPU 板卡本身资源地址以及其它板卡地址发生冲突。

2)获取 I/O 端口访问权限。由于 QNX 对 I/O 端口提供保护,一般用户线程不能直接对 I/O 进行操作,需要调用函数 ThreadCtl(_NTO_TCTL_IO)以确保资源管理器具有访问 I/O地址的权限。

3)I/O 端口地址的映射。由于 QNX 支持虚拟内存管理,尽管取得访问权限,还需要对端口进行映射(调用函数 mmap_device_io( ))后,才可以对 I/O 地址进行操作。

3.3 消息处理函数的编写

客户对设备的具体操作是通过调用文件操作函数间接向驱动程序发送消息,由资源管理器接收并根据消息类型来调用相应消息处理函数来完成的。一般来讲设备操作分为设备的打开与关闭以及设备的读写与控制操作两种情况,每种操作均对应相应的消息处理函数。

1)设备的打开与关闭。客户程序可以调用打开和关闭(如 open()、close())函数来实现。当客户调用这些函数时,资源管理器会得到不同的连接消息。由于这些操作不会对硬件地址进行操作,所以一般采用默认的消息处理函数来实现设备文件的打开与关闭。

2)设备的读写与控制操作。其过程同设备打开与关闭基本一致。不同的是,客户调用读写与控制函数时(如 read()、write()或 devctl()函数),资源管理器得到的是 I/O 消息。因为读写与控制消息处理函数中会含有硬件地址的操作,所以消息处理函数一般由客户自己定义。

3.4 数据的返回

完成对硬件设备的操作后,还需要将信息返回给客户程序。从消息处理函数返回的方法有很多,主要分为无数据状态的返回和带有数据的返回。

1)状态的返回。状态可以简单的分为错误和正确两种情况。为了通知客户程序调用函数出现错误,只需简单的调用函数 return (ENOMEM) 即可实现。如果返回成功的话调用 return (EOK)或者 return (_RESMGR_NPARTS(0))即可。

2)数据的返回。数据的返回有多种方法,最常用的是采用分别指向数据头和数据缓冲区的两个 IOV 数组返回和采用一个包含数据的缓冲区返回两种方法。

(下转第 4 页)

3 系统实现的功能

本系统分四个主监控画面显示 6 个车间的设备状态与生产信息，分别是车身车间、涂装车间、总装车间和其它车间（发动机车间、树脂车间、冲压车间），这四个主监控画面分别显示在四个客户机上，客户机通过网络途径访问服务器的 WinCC 工程。另外，系统提供报表管理、报警记录功能、基础信息维护和监控界面的中英文切换。

系统总装车间监控画面如图 3。

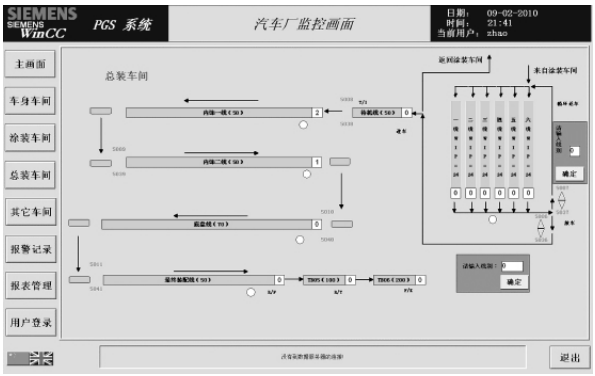

图 3 总装车间监控界面

3.1 区域车辆信息的查询

本系统主要是要实现各个车间中每个区域的车辆信息的查询，车辆信息通过 RFID 系统存储在 SQL 数据库里，WinCC 读取 SQL 数据库里的信息，然后通过界面显示到用户面前。查询信息包括：区域车查询、实绩查询、落后车辆信息查询。

区域车查询：当前时间在这一区域内的所有的车辆信息。

实绩查询：用户输入起止时间，在这一时间段内区域的车辆信息。

落后车查询：在某区域内停留时间超过预设值的车辆信息。

3.2 报警记录

各个车间的设备和通信状态在监控界面实时显示，设备故障以及报警信息都能通过 WinCC Alarm Control 控件查询，还可以查询故障的历史记录和信息，并且可以定时打印故障报表。

3.3 报表管理

WinCC 通过 WinCC Web Browser 控件访问数据服务器上的报表，以 IE 浏览器的方式呈现在用户面前。利用报表系统可以对系统的开启时间、停线时间、RFID 系统中的 READER 读取率等进行保存和打印。

4 结束语

本系统操作简单易懂，易于修改 WinCC 与数据库的连接参数以及工艺参数，HMI 界面友好、功能完善，能够满足公司的监控要求。

参考文献

[1]苏昆哲.深入浅出西门子 WinCC V6[M].北京：北京航空航天大学出版社，2004

[收稿日期：2010.9.11]

（上接第 2 页）

4 设备驱动程序结构示例

本示例中，假设设备共有两种操作：读端口和写端口。客户程序中采用 devctl()函数实现与硬件设备之间的通信。在 QNX 下 devctl()函数是客户程序与资源管理器通信的通用机制。客户可以向资源管理器发送数据、或从资源管理器接收数据等。

在编写资源管理器函数之前，需自定义读写端口命令。此处设备有两个操作，所以首先定义两个命令：

```
#define MY_CMD_CODE 1
#define MY_DEV_READ _DIOF ( DCMD_MISC, MY_CMD_CODE+0 ,int)
#define MY_DEV_WRITE _DIOF ( DCMD_MISC, MY_CMD_CODE+1 ,int)
```

在实际开发中，所定义的命令以及数据类型定义在头文件中，且资源管理器函数和客户程序均要包含此头文件。

然后，定义 devctl 消息处理函数，根据不同的消息类型完成相应硬件操作。结构大致如下：

```
{
    定义客户与资源管理器交换数据类型；
    处理客户程序发送的消息；
    switch(消息类型)
{
case MY_DEV_READ：
读端口；break；
case MY_DEV_WRITE：
写端口；break；
}
返回数据和状态；
……
}
```

最后，定义主函数。主要完成资源管理器以及硬件设备的初始化，结构大致如下：

```
main()
{
初始化资源管理器；
初始化硬件设备；
注册路径名；
循环等待消息；
……
}
```

编译成功之后，运行资源管理器结构的设备驱动程序。在客户端直接调用 devctl()函数就可以实现对设备端口的读写操作。

5 结束语

采用资源管理器模式编写的 QNX 设备驱动程序与在其它操作系统下的驱动不同，它只作为一个用户级的进程独立运行，可以单独编写、调试，做到了与内核无关，也可以根据客户要求任意的启动、停止以及添加、删减，体现了 QNX 模块化程度高和良好的裁剪性等优点。更为重要的是，硬件设备作为文件的形式存在，客户可以通过 POSIX/Neutrino 函数访问，做到了应用程序的设备无关性，保证了移植的方便性和系统的可靠性。

参考文献

[1]QNX Neutrino RTOS，System Architecture. QNX Software Systems[DB/OL].http://www.qnx.com,2010

[2]汤子瀛，哲凤屏，汤小丹.计算机操作系统[M].西安：西安电子科技大学出版社，2006：144-168

[3]Rob Krten,Getting Started with QNX Neutrino: A Guide for Real time Programmers[G]. QNX Software Systems International Corporation, 2009, 82-97. 191-266

[4]黄峰，单家方，匡光力.QNX 系统下 PXI 多功能数据采集卡驱动程序开发[J].计算机技术，化工自动化及仪表，2005

[5]郑胜.基于 PC104 主板的嵌入式数据采集系统的研制[D].西安：西北工业大学，2002

[收稿日期：2010.6.23]