eCos 设备驱动程序设计分析

Analysis of eCos Device Driver Design

西北工业大学计算机科学与工程系 秦怀峰 施笑安 周兴社 谷建华 (西安 710072)

摘 要: 普及计算的发展要求新型的嵌入式操作系统能够支持更多的设备,如何高效、快捷地构造这些网络设备的驱动程序,对嵌入式操作系统的实际应用起着决定性的作用。eCos 这种嵌入式可配置操作系统层次分明、接口合理,便于设备管理和设备驱动的开发,文章在对 eCos 设备驱动程序体系结构分析的基础上,重点讨论了 eCos 设备驱动程序设计编写的要点及其工作的基本原理,并给出了具体的应用实例。

关键词: eCos,设备驱动,内核,中断,宏

1 引言

随着网络接入方式的不断扩展和网络应用模式的日益丰富,'无处不有的网络, 无所不在的计算 '所体现的普及计算日益成为未来计算机技术和应用发展的主要方向之一。普及计算进一步强调以人为本的宗旨, 作为普及计算发展的主要推动力, 嵌入式系统的应用将不再仅限于工业控制系统,要真正实现计算机围着人转、计算能力无处不在、计算工具随身携带、信息资源唾手可得的目标,嵌入式操作系统要能够支持多种多样的设备,这给嵌入式操作系统的设计带来了极大的挑战。

在嵌入式操作系统中,设备驱动程序占到代码量的一半多,同时,嵌入式系统要支持的设备多种多样,因此如何高效、快捷地构造嵌入式设备驱动程序,对嵌入式操作系统的实际应用有重要意义。

eCos (embedded Configurable operating system), 即嵌入式可配置操作系统, 是 RedHat 公司在嵌入式领域的关键产品。eCos 开放源码,相对于价格昂贵的专用嵌入式操作系统, 它具有很大的优势,因此正在吸引越来越多的嵌入式系统开发者。eCos 对嵌入式应用具有良好的支持, 该操作系统层次分明,接口合理,专门设计了便于设备驱动管理和开发的 I/O 包和 DEV 包, 开发人员可以方便地将其开发的程序加入到 eCos 中,和别的系统组件一起进行配置。

设备驱动程序的编写是操作系统开发或移植中最困难的任务之一,本文在对 eCos 设备驱动程序体系结构分析的基础上,着重阐述了 eCos 设备驱动程序的编写,希望能够通过对 eCos 这种典型的开放源码操作系统设备驱动程序构造的分析,使我们能更好地掌握构造嵌入式操作系统设备驱动程序的基本原理, 以广泛应用在各种的嵌入式设备中。

收稿日期: 2003-01-03

基金项目: 十五国防基础研究基金资助项目(J14000B005)

2 eCos 设备驱动程序体系结构

操作系统的设备驱动程序通常包含以下内容:

(1) 提供一些基本的 I/O 函数。它们负责完成以下工作 :初始化和配置设备、从设备收发数据、控制设备、处理设备中断等。

(2) 向内核注册设备。

(3) 调用系统函数,进行设备管理。操作系统内核应提供函数支持驱动程序的同步、计时、内存管理、缓冲区管理、设备名空间及资源管理等

图 1 为 eCos 的体系结构示意,作为一个嵌入式操作系统, 在为应用提供设备操作接口时,eCos 采用了较为灵活的方式。eCos 为应用程序提供了一组统一的 API 用于进行 I/O 操作; 但是,eCos 也允许应用程序绕过统一的设备驱动程序接口,直接访问硬件。

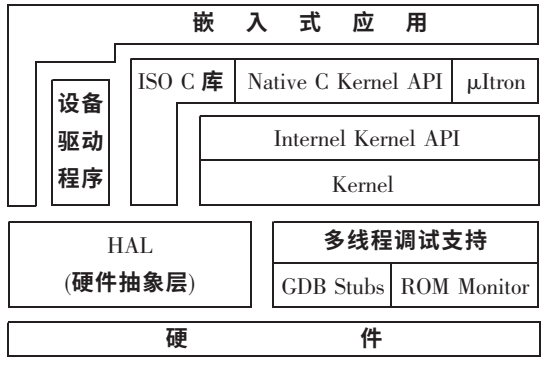

图 1 eCos 体系结构

使用统一的 API 进行 I/O 操作,可以对应用程序屏蔽设备的差别,使应用程序开发者忽略设备 I/O 操作的细节,专心进行应用程序开发,这一方面有利于开发过程中分工的细化,另一方面也有利于增强代码的可移植性和可维护性, 从而提高开发效率,缩短开发周期。但是,通用往往以效率为代价,统一的 API 增加了系统调用开销,延长了系统处理设备 I/O 的时间, 对于一些简单的嵌入式设备,在

应用程序中直接读写设备端口可能会更高效;统一的 API 在处理一些特殊设备时,缺乏灵活性,例如对显示卡的读写操作和对鼠标的读写操作完全不一样,使用统一的 API 为操作系统增加图形显示支持就显得极为笨拙。

eCos 设备驱动由 I/O 包和 DEV 包组成 (包是 eCos 的模块化代码,是 eCos 进行剪裁配置的单位),它们与系统其余部分的相互关系如图 2 所示。

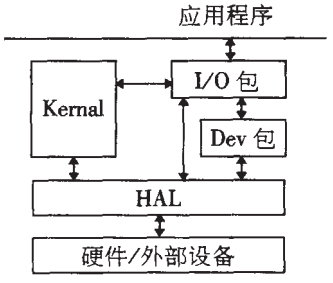

图 2　eCos 设备驱动体系结构

(1) 应用程序:使用 eCos 设备驱动的应用程序通过 I/O API 访问外设。

(2) eCos 内核:操作系统的核心,向设备驱动提供调度、时钟、同步、中断等内核服务的支持。但是,在某些嵌入式应用中,部分内核服务并不是必需的。为了减少代码空间,在配置 eCos 时可以选择将 kernel 包裁剪掉,这时,设备驱动所必需的内核服务由 HAL(Hardware Abstraction Layer),即硬件抽象层提供。

(3) HAL:HAL 包含所有与平台相关的代码,如上下文切换和 I/O 寄存器访问等。HAL 处于 eCos 核心代码的最底层,它直接控制和访问硬件,并通过宏向其它与机器无关的代码提供服务。当操作系统配置不包含内核的时候,由 HAL 向设备驱动提供必需的同步与中断服务。

(4) I/O 包:I/O 包提供抽象的设备 I/O 操作支持。I/O 包为应用程序提供 I/O API,应用程序访问设备时使用逻辑设备名,每个设备都对应一个唯一的逻辑设备名。eCos 设备驱动支持分层的结构模型,即一个设备可以建立在另一个设备之上。这样,一个物理设备就可能对应多个逻辑设备,如"/dev/ser0"和"/dev/tty0"分别为建立在串口 1 上的两个逻辑设备。

(5) DEV 包:DEV 包提供设备 I/O 操作的底层实现,直接操作硬件。I/O 包和 DEV 包之间的界限并不是很严格,设备在 DEV 包中的代码甚至可以完全上移到 I/O 包中实现。但是在驱动程序设计时合理地划分 I/O 包代码和 DEV 包代码,可以增强代码的可重用性。例如,如果要在多个体系平台上为同一种设备(如串口)开发设备驱动程序,那么,将所有对逻辑设备的操作放在 I/O 包中,而只在 DEV 包中编写具体平台的 I/O 操作实现代码,就可以大大减少开发工作量。

3　eCos 设备驱动程序的编写

eCos 的设备可基本分为字符设备、块设备和网络设备三种。目前 eCos 的字符设备主要包括鼠标、键盘、串口等;块设备包括硬盘、闪存等。网络设备在 eCos 里做专门的处理,eCos 支持的网络协议和驱动程序之间有专门定义的数据结构传递数据。下面以字符设备驱动实现的一般过程为例,对编写 eCos 设备驱动程序的几个关键问题进行阐述。

3.1　I/O API

为了统一访问界面,设备的 I/O 操作在各种操作系统中均表现为一组特殊的预定义文件,以便提供统一的 I/O API,设备驱动程序的编写实际上就是这些预定义的文件在具体设备上的实现。I/O 包提供的设备 I/O API,主要对设备进行读、写、设置操作,表 1 是 5 个基本的设备 I/O API,除了 cyg_io_lookup(),其他的 API 调用都需要一个设备句柄。API 调用返回 Cyg_ErrNo 类型的值,如果调用出错,返回值为负,其绝对值为错误类型;唯一合法的返回值是 ENOERR。

表 1　主要的 I/O API

名　称	功　能
cyg_io_lookup()	查找逻辑设备名对应的设备句柄。如果找到指定的设备,则在句柄指针中返回设备句柄。其它的 API 利用该句柄访问设备。
cyg_io_read()	从设备接收数据。参数 buf 为接收数据的缓冲区,可接收的数据量由参数 len 指定,实际接收的数据量也由 len 返回。
cyg_io_write()	向设备发送数据。参数 buf 为待发送数据的缓冲区,待发送的数据量由参数 len 指定,实际发送的数据量也由 len 返回。
cyg_io_get_config()	获得设备的运行时状态信息。检索的信息类型由键值(key)参数指定,检索信息由 buf 参数指定的缓冲区返回。
cyg_io_set_config()	改变设备的运行时状态。修改的信息类型由键值参数指定,数据通过 buf 参数指定的缓冲区传送。

3.2　内核接口

eCos 设备驱动程序通过内核 API 获取内核服务,当操作系统配置不包含内核的时候,由 HAL 提

供类似的 API。这些 API 主要提供必需的中断与同步服务。

eCos 为设备驱动程序提供一种三层中断模型，该模型包括 ISR(中断服务例程)、DSR(延迟服务例程)和线程。ISR 响应硬件中断，DSR 响应 ISR 的请求，而线程是驱动程序的客户程序。硬件中断以几乎不受任何打断的速度传递给 ISR，ISR 可以操作硬件，但只允许调用有限的内核 API。当 ISR 返回时，它可以要求它的 DSR 被调度执行。在绝大多数时候，DSR 总在 ISR 之后立即执行。但如果当前线程正在调度程序中执行，DSR 将被推迟到线程结束。DSR 允许调用比 ISR 更多的内核 API，尤其是 DSR 能够调用内核 API 唤醒正在等待的线程。线程能够调用所有的内核 API，特别地，线程允许等待互斥信号和条件变量。

与三层中断模型对应，eCos 支持三种不同级别的同步机制：

(1) ISR 的同步机制：该机制在临界区禁止中断以阻止 ISR 运行，在多处理机系统中，还要求旋转锁 (spinlock)。ISR 的同步机制通过内核 API cyg_drv_isr_lock()和 cyg_drv_isr_unlock()实现，这种机制应该尽量少用且每次只持续很短的时间。

(2) DSR 的同步机制：该机制在内核中实现，通过在临界区锁调度来禁止 DSR 的执行。DSR 的同步机制通过 API cyg_drv_dsr_lock()和 cyg_drv_dsr_unlock()实现。和 ISR 的同步机制一样，这种机制也应该尽量少用。

(3) 线程的同步机制：该机制用互斥信号和条件变量实现。DSR 可以给条件变量发信号，但只有线程可以给互斥信号上锁和允许等待条件变量。

大多数的设备都提供中断方式的 I/O 操作，为设备编写中断处理程序时，首先要在驱动程序中定义 ISR 和 DSR，然后再调用相关的中断内核 API 将中断对象绑定到相应的中断向量上。根据 eCos 的中断与同步机制，使用中断的 eCos 设备驱动程序可以选择以下三种中断处理模型的一种构造。

第一种模型将所有的设备处理放在 ISR 中完成。当 ISR 被执行时，ISR 直接操作设备硬件、访问内存中待传送的数据。ISR 结束时也可以调用它的 DSR，但 DSR 除了调用内核 API 唤醒线程之外不做任何事情。

第二种模型将对设备的处理推迟到 DSR。ISR 仅仅屏蔽该中断的进一步传递，然后就调用内核 API 使能其它中断，通知操作系统调度它的 DSR 运行。其余的大多数设备处理在 DSR 中完成。

第三种模型将设备处理进一步推迟到线程。ISR 的行为和第二种模型完全一样，而 DSR 仅调用内核 API 唤醒线程，由线程执行所有的设备处理。

中断处理模型的选择应根据设备的特点和嵌入式应用的需要而定，不同的设备可以选择不同的处理模型。第一种模型适用于中断处理操作较简单而系统又需要及时处理的设备；第二种模型少量增加了中断处理迟延，但获得了更灵活的同步机制，适用于多数的设备；如果要在设备中断处理中进行复杂操作，例如，要在鼠标中断到来时分辨不同的事件(移动、单击、双击等)，第三种模型就比较合适。

3.3 两个重要的宏

我们知道，Linux 使用文件操作表来支持对设备的 I/O 操作，文件操作表的每一个域都对应着一个 API，编写 Linux 设备驱动程序就是用实际的操作函数填充文件操作表的各个域。eCos 用设备表(DevTable)和设备 I/O 表(DevIOTable)管理设备，每个设备的设备表和设备 I/O 表分别由 DEVTAB_ENTRY 和 DEVIO_TABLE 两个宏生成，和 Linux 类似，eCos 设备驱动的实现就是用实际的 I/O 操作函数填充这两个宏。

3.3.1 DEVTAB_ENTRY 宏

设备表项由宏 DEVTAB_ENTRY 创建，每次使用 DEVTAB_ENTRY 宏添加一个设备表项。DEVTAB_ENTRY 宏的定义为：

DEVTAB_ENTRY(l, name, dep_name, handlers, init, lookup, priv)

l 为设备表项的“标签”，用于在符号表中标识该设备表项。name 为设备的逻辑设备名，可以是任何有意义的字符串，一般以设备名前加一个 dev 前缀命名，例如/dev/com1。dep_name 用于分层设备，为该设备所依赖设备的逻辑设备名，如果该设备不依赖任何设备，该域为 NULL。

handlers 为指向设备 I/O 表的指针，它连接了设备表项和设备 I/O 表。设备 I/O 表中包含了设备 I/O 操作函数的入口地址，这些函数直接操作物理设备，通过 handlers 域，I/O 包的各个 I/O API 可以调用这些函数完成具体的 I/O 操作。因此 handlers 也是连接逻辑设备和物理设备的桥梁。

init 是设备的初始化函数，它负责初始化硬件和绑定中断。如果使用 3.2 节中的第三种中断模型构造设备驱动，可以在 init 中创建线程。lookup 为应用程序调用 cyg_io_lookup()时设备的处理函数，它

实际上提供了设备‘打开’的操作，在应用程序使用设备前，确保设备为可用状态，并执行进行设备复位，清理缓冲区的操作。

priv 为私有数据。priv 是 eCos 设备驱动程序中一个很有用的域。设备表项中的其它域都有确定的类型，只有 priv 为 void 类型的指针。不同的设备具有不同的特性，管理设备需要的信息也因而不尽相同，由于 priv 为 void 类型的指针，因此它可以携带任何类型的指针，这使得设备表项具有了灵活的可扩展性，可以保存更丰富的设备相关信息。例如，鼠标驱动程序需要保存鼠标的当前坐标值和鼠标允许移动的范围，除此之外，还要保存鼠标的中断向量、中断处理句柄等必要的信息。这时，可以考虑为鼠标驱动程序定义一个数据结构，专门保存上述信息，在使用 DEVTAB_ENTRY 宏时，将这个数据结构作为一个指针传递给 priv，这就扩展了鼠标的设备表项，方便了设备管理。

3.3.2　DEVIO_TABLE 宏

设备表项由 DEVIO_TABLE 宏创建，每次使用 DEVIO_TABLE 宏添加一个设备 I/O 表。DEVIO_TABLE 宏的定义为：

DEVIO_TABLE(l, write, read, get_config, set_config)

l 为设备 I/O 表的‘标签’。write、read、get_config 和 set_config 分别为 4 个读、写、设置 I/O API 的回调函数入口地址。对于设备不提供的操作，相应的域为 NULL。

综上所述，eCos 设备驱动程序的编写步骤可以总结为以下步骤：

(1) 在深入了解设备硬件特性基础上，选择适当的中断模型，定义保存设备专门信息的数据结构。

(2) 调用 DEVTAB_ENTRY 宏和 DEVIO_TABLE 宏向系统注册设备。

(3) 为 init、lookup、write、read、get_config 和 set_config 分别编写设备相关的代码；如果设备使用了中断，根据所选择的中断处理模型，为设备编写中断处理函数。

(4) 调试。

3.4　设备驱动的工作原理

在操作系统配置时，如果设备被选择包含到目标操作系统中，其对应的设备表项在操作系统编译时就会作为内核符号表的一部分被编译到目标代码中去。因此 eCos 不支持设备的动态安装，应用程序可以操作的设备在操作系统配置时就已确定了。图 3 为 eCos 设备驱动程序各个组成部分的相互调

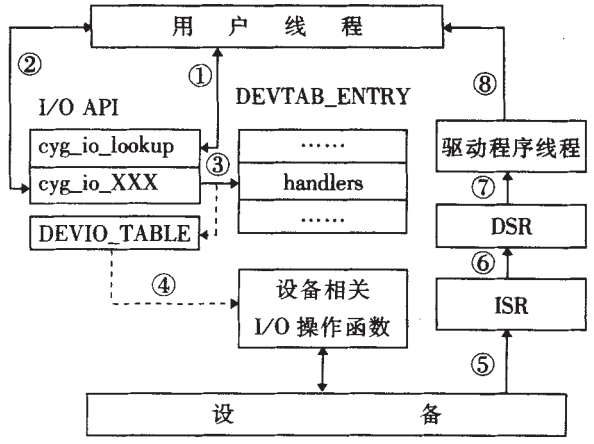

图 3　eCos 设备驱动调用关系

用关系，下面结合图 3 对 eCos 设备驱动的工作原理作简要分析。

(1) 初始化

eCos 在系统初始化时初始化所有的设备。所有设备的设备表项从内核符号表 _DEVTAB_[0]开始，到 _DEVTAB_END 结束。eCos 初始化时，遍历所有设备的设备表，调用设备的 init 函数，初始化设备。设备的 init 函数允许返回错误，这时设备被设置为‘离线’状态且所有对该设备的 I/O 请求都被认为是错误的。将来应用程序操作设备时，将再次尝试初始化设备，但此工作由设备的 lookup 函数完成。

(2) 打开设备

应用程序操作任何设备之前都要先打开设备，打开设备的操作由 API cyg_io_lookup()完成。当应用程序以某设备的逻辑名调用 cyg_io_lookup()时，操作系统在内核符号表中查找设备的设备表项 (图 3①)，如果查找到该设备，则调用设备的 lookup 函数，让设备变为‘在线’状态，允许 I/O 操作进行。对于分层设备，系统还会依次用底层设备的逻辑设备名调用 cyg_io_lookup()。应用程序通过调用 cyg_io_lookup()获得一个设备句柄，设备句柄其实就是指向设备表项的指针。

(3) 操作设备

应用程序以设备句柄为参数调用 I/O API 操作设备(图 3②)。I/O API 将设备句柄作强制类型转换展开为设备表项，通过设备表项的 handlers 域(图 3③)，I/O API 再调用设备相关的 I/O 操作函数完成具体的 I/O 操作(图 3④)。

(4) 中断处理

对于使用中断的设备驱动，根据所选择的中断处理模型不同，当中断产生时，设备驱动分别在 ISR

级(图 3⑤)、DSR 级(图 3⑥)和线程级(图 3⑦)处理中断请求,线程级的中断处理可以进行根据需要和用户线程同步(图 3⑧)。

3.5 应用

结合项目的具体要求,我们先后为 eCos 编写了 PS/2 鼠标、串口鼠标、图像采集卡、专用图像处理加速卡等设备的驱动程序,并成功为 eCos 移植了 tvga 库,为 eCos 增加了图形显示支持,图 4 为部分设备配置到 eCos 中的一个截图。

图 4 eCos 设备驱动配置

驱动程序的设计开发是嵌入式应用开发中比较复杂的一个环节,对开发者有较高的要求,开发者需要同时掌握软硬件方面的相关知识,才能设计出高质量的驱动程序。驱动程序的设计不应拘泥于特定的模式,而应根据应用需要和设备的特性灵活选择适当的模式。

4 结束语

嵌入式计算将是计算机技术发展最重要的领域,也将是最广阔的领域。目前,专用的实时嵌入式系统价格昂贵,开放源码的 eCos 对发展自主的嵌入式操作系统很有借鉴意义,希望本文对 eCos 设备驱动程序设计的介绍,能有助于嵌入式系统开发者更好地掌握构造嵌入式操作系统设备驱动程序的基本原理,以推动相关领域研究的发展。

参考文献

[1] Alessandro Rubini 著, Lisoleg 译. Linux 设备驱动程序. 中国电力出版社, 2000, 50~52.

[2] Rick Grehan, Robert Moote, Ingo Cyliax 著, 许汝峰译. 32 位嵌入式系统编程. 中国电力出版社, 2001, 158~164.

[3] Micbael Barr 著. 于志宏译. C/C++嵌入式系统编程. 中国电力出版社, 2001, 97~100.

[4] eCos Reference Manual. Red Hat Inc, 2000.

[5] Bart Veer, John Dallaway. The eCos Component Writer's Guide. Red Hat Inc, 2001.

[6] Anthony J Massa. eCos Porting Guide. EmbeddedSystem, 2001.

[7] Gary Thomas. eCos: An Operating System for Embedded Systems. Dr. Dobb's Journal, 2000.

QIN Huai-feng, SHI Xiao-an, ZHOU Xing-she, GU Jian-hua
(Computer Science Department, Northwest Polytechnical University, Xi'an 710072)

Abstract: The development of Pervasive Computing demand more devices be supported by the embedded operating system. Therefore, how to program the device driver rapidly and efficiently is becoming an critical problem to the success of embedded operating system. As a configurable embedded operating system, eCos has well designed architecture and interface, which can ease the programming work of device driver. Based on the analysis of eCos's device driver architecture, this article make detailed discussion on some key problems of device driver design and illustrate the working principle of device driver. At the end of the article, we also give some device driver instances we designed for eCos.

Key words: eCos, Device driver, Kernel, Interrupt, Macro

秦怀峰 男,(1976–),博士研究生。主要研究方向为网络化嵌入式计算、嵌入式操作系统等。

施笑安 男,博士研究生。主要研究方向为网络化嵌入式计算、嵌入式操作系统等。

周兴社 男,博士生导师,教授。主要研究方向为网络与分布计算。

谷建华 男,博士,副教授。