基于μCOS-II的USB主机系统①

金　敏，金京林，张　成

(华南师范大学 计算机学院，广州 510631)

摘　要：USB(Universal Serial Bus)是目前应用非常广泛的一种总线形式。其即插即用、热插拔、接口体积小巧等优点给计算机外设连接技术带来重大变革。μCOS-II 是一个源码公开、完整的、可移植、可固化、可裁剪的嵌入式实时操作系统，可以方便地移植到多种微处理器上[1]。以在新太科技实际工作中的 LPC2378 读卡器为例，详细介绍μCOS-II 中 USB 主机系统的设计。

关键词：USB；嵌入式；μCOS-II；读卡器；USB 主机系统

USB Host System Based on μCOS-II

JIN Min, JIN Jing-Lin, ZHANG Cheng

(School of Computer, South China Normal University, Guangzhou 510631, China)

Abstract: USB (Universal Serial Bus) is a far widely used bus. It has many advantages, such as plug and play, hot plug and small volume of the interface, which have hugely changed the development of the technology of connecting computers to external equipments. μCOS-II is an open-source, complete, portable, ROMable, well custom real-time embedded operating system, which can be easily ported to a variety of microprocessor. The paper uses the LPC2378 reader card which I had used in the SUNTEK as an example to introduce the design of USB host system in μCOS-II.

Key words: USB; embedded; μCOS-II; reader card; USB host system

μC/OS-II是美国学者Lacrosse设计的一个优秀的嵌入式实时操作系统，其代码绝大部分用 ANSIC 语言编写，可用于 8 位、16 位、32 位、甚至 64 位微处理器、微控制器、数字信号处理器等，具有操作系统最基本最核心的功能，非常适于在小型系统和片上系统(SOC)中使用[2]。USB 为个人电脑与嵌入式设备之间的连接提供了一种标准化、单一化的接口，其高效性和可靠性使得它已经成为嵌入式系统的首选接口。此 LPC2378 读卡器具有卡票检测、消费扣钱、系统升级、下发黑名单、在线充值、余额查询等功能，但这些功能的实现都依赖于上位机的请求，业务应用模块只有在获得相应的请求后才能进行相应的处理并将处理结果返回给上位机。而 USB 主机系统就是起衔接上位机和业务应用模块的功能，接收上位机请求以及将业务应用模块的结果返回给上位机。

1　构建μC/OS-II系统环境

1.1 移植μC/OS-II到LPC2378开发板

嵌入式操作系统作为大多数嵌入式应用系统的软件平台，它管理着系统的资源，为应用软件提供各种必要的服务。在嵌入式应用系统中使用嵌入式系统，可以提升嵌入式应用系统的开发效率，但是在得到嵌入式操作系统提供服务之前，关键是要将嵌入式操作系统移植到目标板上。

移植条件

移植μC/OS-II之前需要注意，目标处理器必须满足以下五点要求：

1. 处理器的 C 编译器能产生可重入型代码；

2. 处理器支持中断，并且能产生定时中断(通常为10-100Hz)；

3. 用 C 语言可以开/关中断；

① 收稿时间:2010-12-07;收到修改稿时间:2011-01-09

4. 处理器能支持一定数量的数据存储硬件堆栈(可能是几 KB);

5. 处理器有将堆栈指针以及其他 CPU 寄存器的内容读出并保存到堆栈或内存中去的指令。

LPC2378 系列 ARM7 微控制器可以满第 2、4 和 5 点要求,使用 ADS 的 C 编译器可以满足 1 和 3 点要求[3]。

移植步骤

OS_CPU.H 的移植

在 OS_CPU.H 文件中定义与处理器相关的数据类型,例如 BOOLEAN,INT8U 和 INT8S 等。根据 ADS1.2 编译器的特性定义。在 OS_CPU.H 文件中定义与处理器相关的宏,主要是进出临界区代码 OS_ENTRER_CRITICAL()、OS_EXIT_CRITICAL()。将 OS_ENTRER_CRITICAL()和 OS_EXIT_CRITICAL()定义为软件中断函数,所以还要编写相应的软件中断处理代码(可以在 OS_CPU_C.C 文件中编写)实现开/关中断。同样定义 OS_TASK_SW()为软件中断函数,并编写相应的软件中断处理代码(调用 OS_IntCtxSw 函数)实现任务切换[4]。

OS_CPU_C.C 的移植:

在 OS_CPU_C.C 中需要编写 10 个相关的函数,为:OSTaskStkInit();OSTaskDellHook();OSTaskIdle Hook ();OSTaskTickHook()等函数。其中 9 个系统 Hook 函数可以为空函数,也可以根据用户自己的需要编写相应的操作代码。任务栈结构初始化函数 OSTaskStkInit()必须根据移植时统一定义的任务堆栈结构进行初始化。

OS_CUP_A.ASM 的移植

μC/OS-II 移植要求编写的汇编语言函数为:OSStartHightRdy();OSCtxSw();OSIntCtxSw();OS_TickISR()。当然这些程序不一定非得用汇编,也可以用嵌入式 C 语言来完成。

至此,完成μC/OS-II 在 ARM7 处理器 LPC2378 上代码的移植,其大部分代码与μC/OS-II 在其他 ARM7 处理器上的移植是通用的。

1.2 USB 驱动程序设计

μC/OS-II 提供了多任务实时操作系统的内核。在应用这个操作系统时候,用户通常需要自己编写基于μC/OS-II 的外围器件驱动程序,以使外围器件能在操作系统的协调下更好的为用户服务。为了使软件可移植性强,易维护,采用分层的方法编写 USB 的驱动程序。综合考虑 USB 协议、USB 硬件接线、μC/OS-II 的结构来进行分层,下表所列为 USB 驱动程序分层结构。

文件名	简要说明	相关性
USBHAL.c	硬件抽象层	与硬件相关
USBCI.c	接口命令层	与硬件相关
Chap_9.c	协议层	与 μC/OS-II 相关
Descriptor.c	协议层	与 μC/OS-II 相关
USBDriver.c	驱动层	与 μC/OS-II 相关

1.2.1 USB 硬件抽象层

USB 硬件抽象层的主要任务是对 USB 模块的相关硬件进行配置,是 USB 驱动程序的最底层与具体硬件相关的一层。主要完成的任务:初始化 USB 设备控制器为系统配置 USB 时钟控制器,选择信号映射端口(在 LPC 系列中只有 LPC2378 有此功能),配置电源;配置 USB 设备控制器中断,此系统禁止了同步传输帧中断,使端点处于低优先级中断;以及配置软件控制接连、断开和重新连接 USB 功能的相关寄存器。

1.2.2 USB 命令接口层

USB 命令接口层是在 USB 硬件的角度来描述 USB 的具体功能是独立于操作系统之外的,也是协议层和驱动层实现的基础。USB 命令接口层函数基本是和具体的 USB 寄存器相关的,通过操作寄存器完成相应的功能。

1.2.3 USB 协议层

USB 协议层主要由 Descriptor.c 和 Chap_9.c 文件组成。在 Descriptor.c 定义了各描述符,是在 USB 硬件的基础上描述此读卡器的 USB 模块,分别为:设备描述符、配置描述符、接口描述符和端点描述符。这些描述符也是上位机枚举、识别读卡器 USB 模块的媒介。而 Chap_9.c 就是上位机枚举读卡器 USB 模块时 USB 模块回馈上位机的具体实现,其中大部分函数都是依赖于 USB 命令接口层。

1.2.4 USB 驱动层

USB 驱动层是属于 USB 驱动程序中最上层的是与μC/OS-II 系统联系最紧密的一层。在其他各层的基础上从系统的角度描述了 USB 通信功能,是与操作系统和应用程序直接联系的一层。包括系统启动时初始化 USB 硬件的接口以及等待接收主机枚举过程发送的 SETUP 包等函数的接口。其中 USB 端点的读写函

数 USB_ReadPort(INT8U endp, INT32U eppsize, INT8U buffnums, CTRL_USB *pUsb, INT32U len, INT8U *recbuff, INT16U timeout) 和 USB_WritePort(INT8U endp, INT32U eppsize, INT8U buffnums, CTRL_USB *pUsb, INT8U *sendbuff, INT32U len, INT16U timeout)实现了接收上位机的请求和将处理结果返回给上位机。

以读函数为例描述 USB 接收上位机请求的过程，由函数原型的最后一个参数 timeout 可知，读过程是一种阻塞性的操作，在此系统中是以信号量的方式来实现阻塞型的读操作的。在参数检测无误时调用 USB_WaitEpReady(pUsb, timeout)以获取该端点对应信号量，若获取失败则此端点无数据可读。当上位机发送数据到相应的端点时会产生中断，中断处理程序会判断哪个端点产生了中断，然后发送此端点所对应的信号量，这样 USB_WaitEpReady(pUsb, timeout)就可以获得信号量完成读操作，否则程序会等待 timeout 时间，如果在 timeout 时间内依然获取信号量失败那程序就出错返回。若读取长度大于端点缓冲区的长度的话则一次只能读取端点缓冲区长度数，分多次读取，直到读取规定长度为止。写端点函数发送过程和读端点函数接收过程实现流程大体相似，其中最大的区别就是中断产生的时机不同，接收过程是在数据到达相应端点缓冲区时产生中断，而发送过程是将数据写到相应端点缓冲区之后才产生中断。这样在将数据发往相应缓冲区后再调用 USB_WaitEpReady(pUsb, timeout)，若在此函数中成功获得信号量则说明发生成功。

2 USB系统软件设计

USB 的系统软件是与μC/OS-II 操作系统和业务应用模块紧密关联的。在μC/OS-Ⅱ对 USB 进行初始化时，不但要对 USB 硬件接口初始化，还需要对其相关软件进行初始化，比如：设置中断处理函数，以及单独创建一个 TaskSetup 任务以完成上位机对 USB 系统主机的枚举。中断处理过程采用的是非向量中断的方式，首先由中断状态寄存器的值判断中断产生的原因，然后由不同的原因设置不同的中断处理函数。如果是数据中断话则在相应的中断处理函数中发送对应端点的信号量，这样 USB 驱动程序中读写接口才能成功被调用。TaskSetup 是系统的第一个任务，只有在 TaskSetup 任务中 USB 主机系统被成功枚举后才能进行通信。枚举过程主要是通过 0 号端点的控制传输方式进行的，在此过程中 USB 主机系统接收上位机发送的 Setup 包，然后根据 Setup 包的不同请求进行相应的处理再通过控制端点将结果返回给上位机。在 USB 中 0 号端点为控制端点一共有 2 个分别为输入和输出端点。设备枚举其实就是一个上位机识别 USB 主机系统的过程，标准 USB 枚举过程如下：获取设备描述符、复位、设置地址、再次获取设备描述符、获取配置描述符、获取接口和端点描述符、获取字符串描述符、选择设备配置。枚举成功之后 USB 主机系统处于就绪状态并且配置所有的接口与端点[5]。

在 USB 体系结构中数据的交互是以端点为基本单位的，端点的集合表现为接口，在 USB 协议中一个接口表现为一个功能。USB 协议中规定端点 0 只用于控制传输的，其余端点用于数据传输。本 USB 主机系统的数据传输方式为端点 1 采用中断传输，端点 2 采用批量传输。不论哪种传输方式都是以中断的方式和系统交互的，但在中断处理程序中所做的工作非常少只是发送信号量，真正与数据相关的操作并没有在中断处理程序中。这种中断理念是根据 Linux 操作系统的中断上下文思想而设计的：使中断时间尽量的短，将一些对时间要求不是那么严格的事务延迟在中断之外进行。在 Linux 中把中断处理分为上下两部分，中断处理程序是上部分，收到一个中断后它会立即执行，但只做有严格时限的工作例如对接收到的中断进行应答或硬件复位，能够被允许稍后完成的工作推迟到下半部去。比如网卡：数据包的接收是至关重要的以提高网络吞吐量和传输周期以及避免超时，处理和操作数据包的其他工作被推迟到下半部执行[6]。

3 结语

随着电脑外设和数码产品的不断发展，各种设备之间的互连成为当前需要解决的难题。USB 是现今 PC 领域广泛运用的总线接口技术，在一些嵌入式系统中，人们希望有 USB 的出现，然而和系统其他模块相比，USB 模块显得更加的复杂。本文详细阐述了设计一个 USB 主机系统的过程，综合考虑 USB 协议，USB 硬件连接和μC/OS-II 系统使软件易于维护，移植型强。

(下转第 249 页)

（4）第四步：设置预共享密钥。

在 VPN 拨号连接的属性中设置预共享密钥，预共享密钥要保持两端相同。如果使用证书进行身份验证，就不需要这一步了[7,8]。

网关 2 服务器也做上述步骤的相应配置。如果对端是 windowsXP 远程访问客户端，在 XP 的网络连接管理中新建 VPN 拨号连接，不过也需要安装证书或者设置预共享密钥，另外拨号验证的用户名和密码在 VPN 服务器中必须存在，且该用户允许拨入。

5 三种平台的VPN技术比较

用表 1 来表示几种平台的 VPN 实现的特点及性能比较。

表 1 三种平台的 VPN 技术比较

	Cisco IOS	Linux	Windows
成本/价格	高	低	中
适用环境	企业网边界	防火墙内侧	防火墙内侧
配置方式	命令行	配置文件	图形向导
配置复杂度	难	难	简单
通信性能	强	中	中
安全性	强	强	中

6 结语

本文分析了主要的 VPN 隧道技术的区别，并对 Cisco IOS、Linux、Windows 三种系统平台的 VPN 配置步骤进行详细的解析，以展示平台之间对 VPN 技术的配置特色及异同。其实市面上早已有成熟的专用 VPN 产品，这些硬件产品性能好、配置简便、价格各异。不过很多场合我们并不需要专用的 VPN 产品，而是希望在以前的路由器或服务器上集成 VPN 功能以减少设备，而且希望通用平台的配置更趋于简便和高性能，所以研究通用操作系统平台的 VPN 技术仍然有意义。

参考文献

1 徐家臻,陈莘萌.基于 IPSec 与基于 SSL 的 VPN 的比较与分析.计算机工程与设计,2004,26(4):586−588.
2 操惊雷.VPN 安全性分析与应用.黄冈职业技术学院学报,2002,(4):77−79.
3 Deal R. Cisco VPN 完全配置指南.北京:人民邮电出版社,2007.451−510.
4 丛日权.VPN 构建实战—在 Cisco 路由器之间配置基于 IPSec 的 VPN.网管员世界,2005,(5):110−111.
5 郝占军,党小超.Linux 中基于 IPSec VPN 的教育城域网研究与实现.现代计算机(下半月版), 2009,(8):156−159.
6 唐浪,孟相如,陈莉.基于 Linux 的 IPSec VPN 网关的设计与实现.计算机应用与软件,2008,25(12):138−140.
7 刘黎明,张松娟.Windows 2003 Server 下构建 VPN 网络.网管员世界,2009,(1):37−41.
8 赵婧如,王宣政.基于 Windows2000 路由器到路由器 VPN 的设计与实现.西安邮电学院学报,2005,10(4):131−135.

(上接第 224 页)

参考文献

1 赵欣然.基于 μC/OS-Ⅱ系统的 USB 驱动程序的设计[硕士学位论文].呼和浩特:内蒙古师范大学,2009.
2 Mark S. USB Embedded host controller:for removable mass storage devices. Elektor Electronics, 2004, 23(30): 58−63.
3 张德旭.基于 ARM 的嵌入式 USB 主机系统的研究[硕士学位论文].哈尔滨:哈尔滨理工大学,2009.
4 韩志耕,王健.实时内核 μC/OS-II 在 S3C44BOX 上的移植的研究与实现.计算机工程与设计,2006,27(5):828−831.
5 傅得立.基于 USB2.0 的数据记录回放单元设计[硕士学位论文].成都:中国科学院光电研所,2007.
6 陈莉君.Linux 内核设计与实现.第 2 版.北京:机械工业出版社,2009.60−61.