嵌入式环境中的软件构件化研究

程广河,郝凤琦,张让勇,韩路跃,罗 旋,任绪才
(山东省计算中心,山东 济南 250014)

摘 要:根据嵌入式系统的固有特点,将构件化的思想引入到嵌入式系统中,对嵌入式软件进行了构件化改造。针对底层驱动和操作系统的构件化,可有效提高软件系统的可移植性以及安全性;对应用层软件的构件化,提高了嵌入式软件的可配置性,并减少了软件系统的每个应用的体积。最后以嵌入式通讯协议栈为例,测试了构件化对软件性能的影响,对嵌入式软件的构件化改造有参考意义。

关键词:嵌入式软件;构件化;协议栈;
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2007)09-0139-03

Research of Software Component for Embedded Environment

CHENG Guang-he, HAO Feng-qi, ZHANG Rang-yong, HAN Lu-yue, LUO Xuan, REN Xu-cai
(Shandong Computer Science Center, Jinan 250014, China)

Abstract: Because the existing embedded software has its limitation, aims at these disadvantages and proposes some creative suggestions. By making use of the component idea, succeeds in carrying out special component for devices drivers and some relative functions of operating systems. This method can improve the transplant and configuration characteristics of embedded software efficiently and decrease the bulk of protocol stacks for every application. Finally, makes some relative performance testing on testing system. Test how the component affects the performance of embedded software, and the result is a reference to alteration of embedded software.

Key words: embedded software; component; protocol stack

1 构件化技术

嵌入式系统是针对具体应用的专用系统,一般具有成本敏感性。其硬件和软件都必须进行高效率的设计,量体裁衣、去除冗余,力争在同样的硅片面积上实现更高的性能。传统的嵌入式软件考虑到上述因素,大都尽量压缩代码量,以减少软件的资源需求。从而造成大部分的嵌入式软件结构紧密,可读性、可移植性都比较差。而实际上随着硬件技术的发展,目前嵌入式系统的资源已相对富裕,在满足资源需求的基础上,已可以用一些传统软件的思想来改造嵌入式软件。

基于构件的软件开发(CBSD, component - based software development)方法就是一种可以提高软件复用性的软件开发方法,其中主要的代表有 COM, EJB, CORBA 等,它们的主要构架思想是分布式、封装性、扩展性、复用性的结合[1]。CBSD 是在模块化系统、结构化设计和面向对象技术的基础上发展起来的,它使用预先开发的、经过验证的软件构件,大大节约了软件开发过程中的人力物力投入,提高了软件质量[2]。

与传统的嵌入式软件不同,构件化的嵌入式软件是由一组软件构件构成的,这些构件的一个或者几个组合成一个完整的应用;而且,新的应用也可以使用已有构件,从而提高软件复用性[3]。因此,传统的嵌入式软件提供的是专用的服务,软件与应用是一一对应的,而构件化的嵌入式软件则提供了一种组合式服务,软件复用性、扩展性大大提高。构件化的嵌入式软件结构与传统嵌入式软件结构区别如图 1 所示。

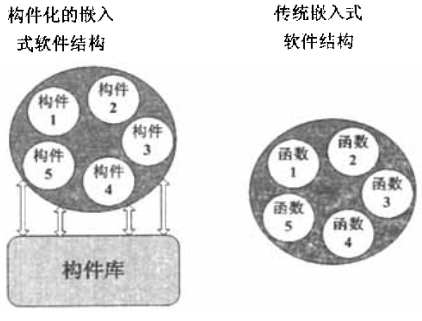

图 1 构件化的嵌入式软件与传统嵌入式软件

传统嵌入式软件是单块式结构,每个应用都由结合比较紧密的一系列函数构成;构件化的嵌入式软件

收稿日期:2006-11-22
作者简介:程广河(1966-),男,山东济宁人,硕士,副研究员,主要研究方向为嵌入式系统、工业控制。

形成一个通用性较强的构件库,运行中的嵌入式软件由构件库中的某些构件组合而成,软件结构比较灵活,而且由于构件可以在运行中调入调出,因此针对某个具体的应用而言,构件化的结构往往比单块式结构更节省系统资源。

2 嵌入式软件的构件化改造

嵌入式系统多种多样,对应的嵌入式软件也具有种类繁多、结构复杂和多样化的特点。按照其功能以及所处层次的不同,可以把嵌入式软件分为以下几种:硬件驱动;嵌入式操作系统;应用程序。在对其进行构件化改造的时候有以下几种方式:按层次构件化、按功能构件化和按应用对象构件化。对于硬件驱动,由于其与底层硬件的关系密切,构件化改造可将硬件驱动中与底层关系密切的部分剥离,将其数据的初处理以及与上层操作系统的接口部分改造成统一的构件,即按照层次进行构件化;对于嵌入式操作系统,应彻底剔除其对底层硬件依赖性,对硬件的操作仅通过驱动构件进行,操作系统只负责任务调度、任务间通信、资源分配等核心功能,并根据功能的分割,将其构件化为任务管理构件、通信管理构件、驱动管理构件等;针对应用程序的构件化,是在一类应用中(比如网络设备),不同的具体功能构件化为专门的构件,针对某个应用,可能是一个或者几个应用构件的组合。

在由上述的几种构件组合成一个嵌入式软件系统时,采用了一种"灵活内核"的设计,操作系统的内核地址区可以看作一种特殊的地址空间,只有经过授权的构件才能被拿到这个空间运行。根据底层硬件设备的不同,可以将一些比较成熟稳定的驱动构件配置成"内核态",而其他一些比较新的硬件设备,其驱动程序可以配置成"用户态",从而在不威胁系统稳定性的基础上尽可能地提高系统的运行效率;针对用户需求的不同,可以将一些"久经考验"的应用构件配置为"内核态",而将一些不太成熟的应用构件配置为"用户态"。整个软件系统架构如图 2 所示。

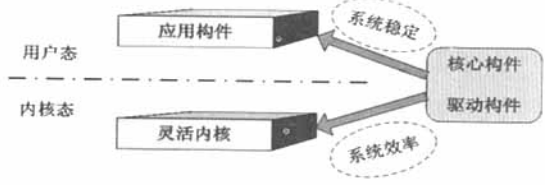

图 2 系统软件架构

2.1 驱动构件

在对驱动程序进行构件化改造后,驱动构件需提供上层访问接口,参考了 Linux 系统设备驱动程序的管理方式,将设备驱动程序分为字符设备和块设备两类,驱动构件的上层接口也只有这两类:字符类(裸设备,无缓冲区)和块设备(有缓冲区)。操作系统中的驱动管理构件通过接口对驱动构件进行加载、硬件资源管理等操作。不同于普通的构件,驱动构件是与硬件紧密相关的,这就涉及到如何为驱动构件分配指定的中断号、与设备相关的缓冲区(内存),这些都由操作系统中的驱动管理构件来完成。

驱动构件可以进行动态地加载和卸载。这使得嵌入式系统中驱动程序的单独升级成为可能,构件化的驱动程序将驱动程序的内部实现细节隐藏,只提供有限的几种接口,保证了驱动程序的修改不会影响使用驱动构件的用户程序。

另外,从面向编程的角度看,构件化的驱动程序抽象化了硬件设备,用户看到的是统一的两种设备:字符设备和块设备。底层的中断管理用户也无需操心,因中断处理程序已经融入到各个设备的驱动构件中。

2.2 核心构件

这部分的工作主要是针对嵌入式操作系统的构件化改造,称之为核心构件。操作系统的最主要功能是任务(构件)管理,其主要功能有以下几个方面:

(1) 根据构件的自描述信息生成构件的运行空间,即用户构件运行所需的内存空间,需要运行或者加载的驱动构件。驱动构件加载所需的缓冲区等。

(2) 构件之间的通信机制,构件的运行状态监控、错误报告等。

(3) 构件的干预机制,可以强行终止错误状态的构件。

操作系统的构件化改造将操作系统中最为核心的部分(任务管理、资源管理、通信、驱动管理)分别构件化,运行的时候置于"内核态",用户程序则位于"用户态",并且构件化的思想使操作系统的接口统一规范,不安全的用户操作很难影响操作系统的运行。这就有效提高了嵌入式操作系统的健壮性。

操作系统的非核心部分,如硬件资源的抽象化(与驱动构件配合),在构件化改造中可以将其置于"用户态",甚至直接放到底层的驱动构件中。如用户经常用到的定时器资源,在底层驱动构件将定时中断包装之后,操作系统部分只需将其抽象化为定时器构件,接口中将定时器参数(比如计时精度)进一步细化即可。

2.3 应用构件

随着嵌入式系统功能的复杂性提高,嵌入式系统的软件的功能也越来越多,而实际运行中,这些功能使用的频繁程度是不同的。利用构件化的思想,就可以将构件动态地调入调出:将应用软件按功能进行构件

化,这些应用构件在核心构件的管理下,动态地调入内存执行,使用频繁的应用构件可能一直位于内存中,随时可以运行,甚至进入"内核态",提高系统运行效率;而使用不频繁的应用构件只有需要运行的时候才调入,从而减少了系统资源的消耗。

2.4 构件化的粒度

嵌入式软件的构件化需要考虑的一个重要问题是构件化的粒度。从粒度上来看,构件的粒度越小,软件划分的越细,构件数量越多;构件粒度越大,软件划分的越粗,构件数量就越少。构件粒度的大小,决定了构件的模块化、信息封装性、局部化的程度,为此必须保证构件的独立性。一旦构件具备良好的独立性,建立在构件之上的应用程序就更容易开发,接口也会简化;独立的模块也比较容易测试和维护,修改工作量小,错误传播范围小。如果粒度过小,虽然构件独立性增强,但是构件的接口就增加了,给构件的组合、构件的管理带来了许多的困难;如果粒度过大,构件的尺度增加,独立性降低,各个构件之间的关联度也会增加,不利于构件的动态替换与更新。

3 构件化对软件性能的影响

文中主要探讨了嵌入式环境中的软件构件化,为了研究构件化对软件性能的影响[4],在实验室现有条件基础上,搭建了相关的测试平台,对一套嵌入式网络设备的软件进行构件化改造,对改造前后的软件性能做了测试。

3.1 测试平台(软件硬件平台)

测试硬件平台如图 3 所示:整个系统通过集线器(HUB)连接起来,嵌入式设备(实际试验中采用的是周立功公司的基于 ARM7 的嵌入式开发板)提供实验所需的硬件资源,设备上运行的软件系统有三个部分:硬件驱动(网卡、flash 等)、嵌入式操作系统、应用程序(TCP/IP 协议栈以及网络程序)。另有 PC 机一台连入系统,由于集线器对所有收到的包都做无条件的广播,所以连入系统的所有设备的通讯对彼此都是可见的,PC 机上运行有 Sniffer 监控软件,可以监控整个系统的通信。对 TCP/IP 协议栈构件化前后的性能进行了对比。服务器端运行有相应的服务器程序,与嵌入式设备完成通信。

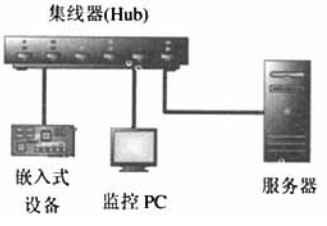

图 3　测试平台结构图

3.2 性能测试

协议栈的构件化会影响协议栈的性能,因为协议被构件化之后,必然会增加协议栈构件调入调出内存的次数,增加数据拷贝的次数,从而影响协议栈性能[5]。表 1 数据列出了不同的构件化粒度下,通过 FTP 协议下载服务器上同一个文件的性能对比,表 1 中的平均下载速度是从服务器端看到的整个下载过程的平均下载速度,总帧数是从监控 PC 上由 Sniffer 程序抓取到的整个通信过程的所有帧的数量。

表 1　构件化对协议栈性能的影响

	平均下载速度(KBPS)	总帧数(个)
未构件化	358.22	7702
粒度 1	390.60	7586
粒度 2	335.65	7788
粒度 3	302.14	7922

表 1 中粒度 1 到 3 构件化的粒度依次减小,其中粒度 1 仅对 FTP 协议做客户端和服务器端粗粒度的构件化,可看出,构件化后的协议栈性能有所提高,这是由于构件化减小了协议栈具体应用的体积(本例中仅用到客户端构件),从而减少了系统资源的消耗,提高了效率;粒度 2 增加了系统构件,各构件用到的系统功能均从系统构件中调用,增加了构件进出内存的次数,协议栈性能稍有下降;粒度 3 则将协议栈进行了全面的构件化,包括核心协议都进行了粒度较小的构件化,可看出协议栈性能有较大下降。由以上分析可看出,构件化对协议栈性能有较大影响,但从 Sniffer 抓取的帧总数上来看,差别并不大,说明构件化并没有使协议的工作流程发生变化,只是增加了协议构件调入调出的次数以及数据拷贝的次数。但通过构件化提高了嵌入式软件的可移植性、可配置性以及可靠性,精简了嵌入式软件的每个应用的尺寸,节省了存储空间。

参考文献:

[1] William, Deshpande S. Integrating CORBA and EJB[EB/OL]. 2001 - 01. URL: http://www.middleware - company.com.

[2] 袁飞云.基于 COTS 构件组装的系统开发[J].微电子学与计算机,2006,23(8):38 - 41.

[3] 古幼朋.嵌入式实时软件的构件化开发技术研究[D].成都:电子科技大学,2005.

[4] Dunkel A. lwIP - a lightweight TCP/IP stack[EB/OL]. 2002 - 10. URL: http://www.sics.se/adam/lwip/.

[5] 甘　泉.通讯协议分解和构件化的协议子模块[D].杭州:浙江大学,2005.