第29卷 第6期 计 算 机 学 报 Vol. 29 No. 6
2006年6月 CHINESE JOURNAL OF COMPUTERS June 2006

一个构件化嵌入式操作系统的精确控制内核

陈文智 谢 铖 石教英
(浙江大学计算机科学与技术学院 杭州 310027)

摘 要 近年来,嵌入式系统的应用数量和复杂度以及程序的规模增长迅速,构件化嵌入式操作系统已经成为研究热点.过去的以抽象接口和封装计算为基础的通用构件体系结构难以满足许多数据流应用(包括多媒体处理和信号处理等)在并发性和实时性等方面的要求.对此设计了一个构件化嵌入式操作系统 Pcanel,其核心是一个基于构件转换控制模型的精确控制核 PCC. PCC 将构件分成静止和执行两种状态.构件计算采用并行与分段执行的方法.同时,采用非阻塞式的调度方法有效解决了优先级倒置的问题.PCC 由事件控制器和任务控制器协同工作,具备高度并发处理能力,从而支持构件技术在嵌入式系统中的应用.

关键词 嵌入式;构件;操作系统;模型驱动
中图法分类号 TP311

A Precise Control Core for Component-Based Embedded Operating System

CHEN Wen-Zhi XIE Cheng SHI Jiao-Ying
(*Institute of Computer Science and Engineering, Zhejiang University, Hangzhou* 310027)

Abstract Embedded applications are proliferating at an amazing rate. The continuous growth of complexity and scale of embedded system motivates the research focus on Component-based Embedded Operating System. However, general component-based architectures that are built on interface of functionality and encapsulation of computation fail to satiety concurrency and real-time issues of data flow applications, such as multimedia processing and signal processing. Pcanel is a novel component-based operating system for embedded systems. The paper describes the kernel, called Precise Control Core (PCC), which is based on a component transition model. PCC partitions states of component into quiescent states and executing states. The computation of concurrent components is recomposed into a sequence of split-phases of transitions. A wait-free synchronization technique is used preventing priority inversion. The collaboration of event controller and task controller of PCC achieves massively concurrency, thus supports using components in embedded systems.

Keywords embedded; component; operating system; model driven

1 引 言

在3C(Computer, Communication, Consumer)合一的产业趋势下,基于数据流的软件应用成为了许多嵌入式操作系统的主要部分,这包括多媒体处理、网络通信、信号处理等.数据流应用往往需要大量的计算资源和功耗,因此嵌入式操作系统应当为

收稿日期:2004-08-03;修改稿收到日期:2006-03-02.本课题得到国家"八六三"高技术研究发展计划项目基金(2002AA1Z2302,2004AA1Z2050)和浙江省重点科研项目基金(2004C21059)资助.陈文智,男,1969年生,博士,副教授,主要研究方向为嵌入式实时系统、分布式计算、多媒体网络和网络存储. E-mail:wzchen@cad.zju.edu.cn. 谢 铖,男,1977年生,博士研究生,主要研究方向为嵌入式系统和计算机系统结构. 石教英,男,1937年生,教授,博士生导师,主要研究领域为分布式并行计算、虚拟现实和计算机辅助设计与图形学.

这类应用作出特别的支持或优化.业界已有的构件化嵌入式操作系统主要把重点放在如何应用构件技术创建操作系统本身,构件之间的交互以方法调用(或远程过程调用)形式为主,较少考虑数据流应用领域的特点,不能满足高效性和实时性等方面的要求.

构件化操作系统 Pcanel[1]用一个模型驱动的构件框架取代基于方法调用的构件模型,更适合于嵌入式系统的数据流应用. Pcanel 将构件之间的交互过程用计算模型(Model of Computation)[2]加以规范化.

构件转换是 Pcanel 的基本工作单元,由构件的计算行为和触发条件组成,构件转换的精确控制核 PCC(Precise Control Core)是核心的构件执行框架. PCC 将计算模型映射为令牌(Token)调度网络,完全控制构件的计算行为和状态变化,支持嵌入式系统的高强度并行和实时处理,维持最小的硬件资源要求. PCC 的事件控制器和任务控制器联合成整个构件连接网络的控制器.

模型驱动的关键实现在于将控制流与构件的计算行为相分离[3],这意味着 PCC 完全控制着构件的执行过程和状态转换,因此一个构件可以重复地在多种不同的上下文环境中执行. PCC 还可进一步对多个互相关联的构件进行优化,比如消除多任务间的上下文切换、应用静态调度等. PCC 对构件接口进行验证,并控制构件连接网络的整体状态,从而做出更好的预测等.与采用通用构件模型的操作系统相比,模型驱动的操作系统能够更好地面向特定应用实现定制.通过强化构件接口的约束条件,构件更容易被组合和验证.由于 PCC 采用计算模型来精确控制构件的执行,因此可用来构造一个可靠的构件系统.

2 构件转换控制模型

本节采用形式化方法描述 PCC 精确控制构件转换的理论模型.

一个框架 F 是一个状态与行为的转换系统 (S,A,T,I),其中,S 是一组状态(states)集合;A 是一个行为(action)集合;T 是一个转换(transition)集合,$T\subseteq S\times A\times S$,对于任一转换 $\tau=(s_1,a,s_2)\in T$,s 和 s' 分别表示 τ 的开始状态和结束状态,简写为 $\tau_{(s_1,s_2)}$. I 是一个初始状态(initial state)集合.

框架 F 的状态 S 反映了框架内部所有构件共享的信息,如时间、计算模型等.比如构件采用堆栈作为传递令牌的队列,那么这些堆栈就是一部分框架状态.

一个构件 C 存在于框架 F 中,用 $(S_C,G_C,A_C,T_C,Q_C,I_C)$ 表示,其中,S_C 是构件状态的一个集合,其中构件通过计算模型进行输入输出通信所改变的状态称为构件的端口,其它则称为构件内部变量;G_C 是框架 F 的一组状态,用于触发构件的行为;框架 A_C 是构件的行为集合;$T_C\subseteq G_C\times S_C\times A_C\times S_C$ 是一个转换集合,对于一个构件转换 $\tau=(G_\tau,s,a,s')$,简写为 $\tau_{(s,s')}$,$G_\tau\subseteq G_C$ 称为该转换 τ 的触发条件;$Q_C\subseteq S_C$ 是一组静止状态;$I_C\subseteq Q_C$ 是一组初始状态.

触发条件 G_τ 由三个部分组成:触发时间 GT、一组输入端口条件、一组输出端口条件.当 G_τ 所有条件都满足时,框架执行构件转换 τ,由 *ReadPort*,*Action*,*WritePort* 三个函数组合实现.如图 1 所示,代表构件主体计算行为的 *Action* 函数改变构件内部状态(从 s 到 s'),它总是发生在 *ReadPort* 函数之后到 *WritePort* 函数之前.

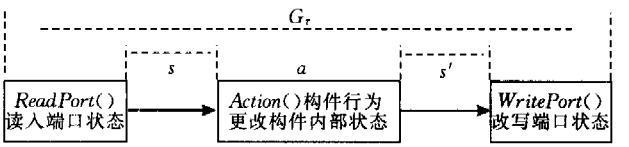


图 1 构件转换的三个阶段

一组构件通过计算模型组合成一个框架,构件与框架进行同步交叉执行.在任何时刻,当框架 F 的状态满足构件 C 的某一个转换 τ 的触发条件 G_τ 时,转换 τ 将执行,当 τ 执行后可能改变框架的状态.通常情况下,框架和构件的执行都不必具备确定性,一个框架的状态可能同时触发多个构件,但是构件行为的执行顺序是不确定的.

一个构件转换 $\tau_{(s,s')}$,从状态 s 转换到状态 s' 可能经过一个或多个串联的转换,用执行路径 $EP_{s,s'}=\{\tau_1,\tau_2,\cdots,\tau_n\}$ 表示,其中 τ_1 的开始状态是 s,τ_n 的结束状态是 s',τ_i 的结束状态是 τ_{i+1} 的开始状态.当 $EP_{s,s'}$ 只能包含一次转换 τ 时,则 τ 是从状态 s 到状态 s' 的原子转化.用 $EP_{s,s'}\lrcorner s''$ 表示 $EP_{s,s'}$ 经过状态 s'',即 $\rightarrow n>1$,$\exists 1\leq i\leq n-1$,$\tau_i\in T_C$,$\tau_{i+1}\in T_C$,使得 s'' 是 τ_i 的结束状态和 τ_{i+1} 的开始状态.当 $EP_{s,s'}$ 对于 S_C 是原子转化时,用 $EP_{s,s'}\lrcorner\Phi_{S_C}$ 表示 $EP_{s,s'}$ 不经过 S_C 中的任何状态.

构件隐藏内部状态和转换,只提供对框架有意义的静止状态 Q_C.当构件处于静止状态时,表示构件进入静止阶段;当构件的触发条件满足开始转换

时，表示构件进入执行阶段. 因此构件在时间刻度上总是进行静止阶段和执行阶段的交替. 构件从静止阶段 q 进入执行阶段的第一个转换 τ_q 称为触发器，表明构件进入一个新的执行阶段，在该阶段结束后，框架的状态通常会被更改.

由于所有构件总是从一个静止阶段进入另一个静止阶段，因此框架很容易控制内部构件的转换过程，在组合构件的静止阶段，内部构件必定处于自己的静止阶段. 从更高层面来看，就可以把这个执行路径抽象为一个对框架 F 有意义的转换 τ，它对于 Q_C 而言是原子转换，从构件内部的串联转换抽象成框架原子转换的过程就是构件转换的组合. 形式上来说，即 $\exists q, q' \in Q_C$，q 和 q' 分别是 τ 的开始状态和结束状态，并且 $\tau \lrcorner \Phi_{Q_C}$. 为了明确 τ 所包含的串联转换路径总是开始和结束于静止状态 (q, q')，用 $PT_{(q,q')}$ 表示，称为对于框架 F 的精确转换. 图 2 演示了构件转换的组合. 如果框架 F 的状态变化允许 $PT_{(q,q')}$ 中所有串联的转换都能够发生，那么 $PT_{q,q'}$ 总是能够实现. 然而，框架的状态变化不仅取决于框架本身，而且也取决于内部所有构件的执行，因此对精确转换的分析可能相当不容易.

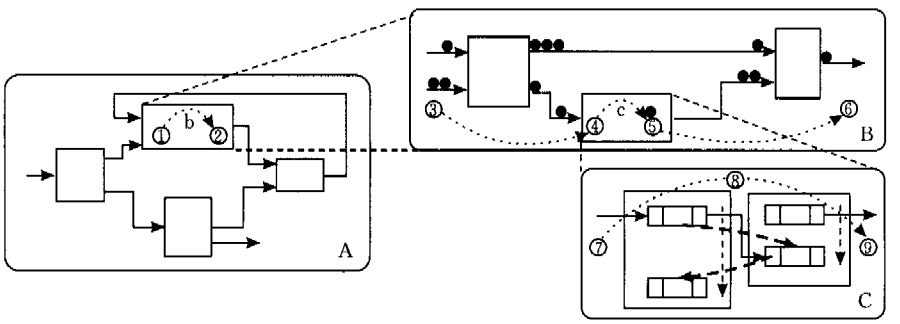


图 2　分级组合的计算模型

如图 2 所示，框架 A 采用 DE 模型，框架 B 采用 SDF 模型(图中的黑点表示端口输入、输出的令牌)，框架 C 采用 CSP 模型，其中框架 C 经过适当扩展封装成框架 B 中的构件 c，框架 B 经过适当扩展封装成框架 A 中的构件 b. 框架 C 的执行路径 $EP_{(7,9)} = \{\tau_{(7,8)}, \tau_{(8,9)}\}$ 被抽象成框架 B 的精确转换 $PT_{(4,5)}$，同样的，框架 A 的精确转换 $PT_{(1,2)}$ 包含了框架 B 的执行路径 $EP_{(3,6)} = \{\tau_{(3,4)}, \tau_{(4,5)}, \tau_{(5,6)}\}$ (以及子构件内部的执行路径).

框架和构件之间的协作是实现精确转换的必要保证. 一个触发器 τ_q 是精确的，当且仅当它可以保证，对于框架所有将来可能的状态，在服从框架实现的计算模型时，从 q 开始的执行过程总能够到达另一个 $q' \in Q_C$. 一个框架是精确的，当且仅当它可以在给定构件的精确触发器时保证所有构件的精确转换. 为了完全实现精确框架，PCC 必须在接受一个构件的精确触发器后，控制框架状态变化，以便维持构件对精确触发器要求的环境. 这个过程要做通用分析非常困难，但是如果框架实现了某种规范的计算模型并且限制构件之间的交互模式，那么分析就比较容易了. 为此，在分析过程中将构件的计算和通信分离，凡是不引起框架状态变化的构件转换，称为内部转换，除此之外的其它转换都称为输入、输出转换. Q_C 的精确转换所包含的执行路径 $EP_{s,s'} = \{\tau_1, \tau_2, \cdots, \tau_n\}$ 中，τ_1 和 τ_n 是输入、输出转换，$\{\tau_2, \tau_3, \cdots, \tau_{n-1}\}$ 均为内部转换.

精确框架并不完全防止死锁，事实上，有可能在某些框架状态下，没有一个触发器能够触发，以至于没有任何一个构件能够执行，然而，它们确实保证，即使在死锁状态下，所有的构件都处于静止状态，因此整个 PCC 系统作一个整体处于可知的静止状态.

3　控制器运行基础

PCC 控制构件运行的基础主要有四个关键部分：基于任务和事件的并行；非阻塞式同步；分段操作；高效的构件间通信. 在此基础上，精确表达和控制构件转换，实现高性能并行和实时处理.

3.1　基于任务和事件的并行

在 PCC 中具有两种并行的来源：任务和事件. 任务是一个延期的计算机制. 他们运行到结束为止，互相之间不抢占. 构件可以递交任务，递交操作立即返回，PCC 将调度该任务延迟到某时间以后执行. 如果时间要求不是很严格的话，构件可以直接激活任务，这包括几乎所有操作，除了低级通信. 为了确保很低的任务执行延迟，单个任务必须很短小，很长

的操作应该跨越多个任务来实现.嵌入式系统中软件的生命周期一般较长,为了保持系统的应激性,通常禁止计算量很重的任务.事件意味着一个任务的完成或者来自环境的事件异常、中断或者时间触发,事件的处理也是运行到结束为止,但是可以抢占任务或者其它事件的执行.PCC 的执行根本上来说是由代表系统内外物理信号的事件驱动的.如图 3 所示,从时刻 t_0 开始,构件 A 的三个任务与构件 B 的三个任务交叉执行.构件 B 的一个任务在时刻 t_3 被事件抢占,在时刻 t_4 恢复执行,构件 B 在时刻 t_5 之前递交新的任务,PCC 在时刻 t_7 调度执行该任务.图中还说明,在时刻 t_6 前后,构件 A 的两个任务连续执行,为保持较轻的计算量,并不进行合并.

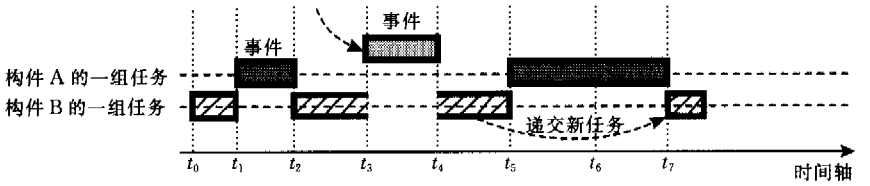


图 3 基于任务和事件的并行

3.2 非阻塞式同步

构件端口是 PCC 中唯一的一个关键区,由于事件可以抢占事件或任务,因此当事件与任务使用相同端口进行令牌通信时,必须保证同步.为了满足事件的硬实时响应要求,中断延时必须足够地小.PCC 采用非阻塞式同步机制控制端口访问,实现完全可抢占性,并有效避免了优先级倒置问题.非阻塞式同步的原理是,关键区维护所有访问线程的上下文环境(context)的队列,当高优先级的线程 B 进入关键区时发现被低优先级的线程 A 占用,则 B 帮助 A 先完成其关键区指令,帮助期间,A 的指令具有与 B 相同的优先级,当 A 完成后,B 继续执行自己的关键区指令.如图 4 所示,优先级从低到高的事件 A,B,C,具有各自的环境.某时刻 A 处于关键区时,被 B 抢占,当 B 运行至关键区入口时,B 的环境保存至关键区队列中,然后切换到 A 的环境中帮助执行 A 的指令,若在此过程中又被更高优先级的 C 抢占,则关键区队列增加为(A,B,C),C 先后帮助完成 A 的指令和 B 的指令,最后恢复执行自己的关键区.

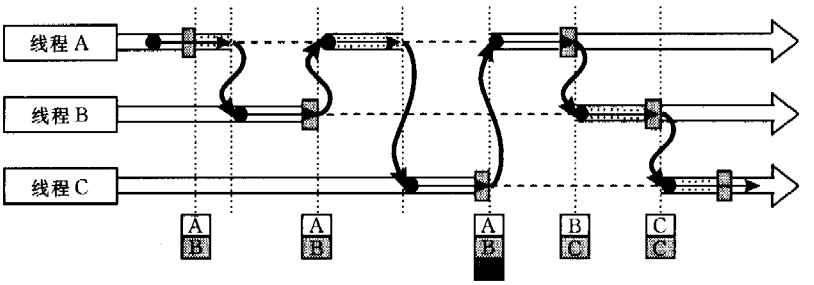


图 4 非阻塞式同步

3.3 分段操作

因为任务之间是不可抢占地执行,所以 PCC 执行的所有操作都具有非阻塞特性.所有可能阻塞的操作都是分阶段的:操作的请求、执行和结束是三个分离的计算阶段.一个典型的分段操作就是构件间通信:构件 A 在某一通信模型下发送消息给构件 B,构件 A 唤醒"发送"命令来初始化一个消息的传送,而实现通信模型的构件框架只负责传送信息,当完成时激发构件 A 的"发送完毕"事件和构件 B 的"新收到"事件.每个构件都只实现执行阶段,通过发送或者接收事件而互相调用.

资源争夺的问题通过明确的拒绝并行请求得到处理.当阻塞于争夺的资源时,在基于线程的并行模型中,线程的堆栈将消耗宝贵的内存,相比之下,PCC 的简单并行模型允许高度并行以及低开销.然而,正如在任何并行系统中一样,并行度和不可决定性可能是复杂的错误来源,包括死锁和数据竞争.

3.4 高效的构件间通信

PCC 支持构件以令牌通信方式进行协作,连接的构件形成一个令牌通信网络.构件均运行在内核地址空间,PCC 加载构件时对接口进行验证,以保证组合系统的安全性,构件提供的服务以端口间令牌通信的形式进行组合,并在部署到目标系统之前实施静态优化,将计算模型的算法和数据结构映射

到实际的处理器指令和内存地址.

较早期的一些构件化操作系统，用进程封装构件的计算，以进程间通信来实现构件协作，效率较低，为了减少对系统整体性能的影响，通常把尽可能多的计算放到一个大构件中实现，因此不能更好地发挥构件组合的优点. PCC 采用细粒度构件的分级组合机制，克服了构件间通信的性能缺陷，使得操作系统能够采用更彻底的构件化设计.

4　控制器架构

4.1　令牌调度网络

传统操作系统的调度器都是基于线程(进程)优先级的，而在构件化操作系统中，所有构件通过分级组合形成一个全局的构件连接网络，令牌是这个网络的流通单元，因此 PCC 的本质是基于令牌调度构件转换. 构件转换建立在状态机模型上，每个状态代表构件转换的一个执行步骤，可能要求一个或多个令牌，代表特定的数据结构或者资源，构件框架所采用的计算模型由相应的令牌管理器实现，控制构件转换使用的令牌，从而达到对状态机的精确表达. 所有的令牌管理器构成一个全局的令牌调度网络. PCC 以基本的令牌通信机制协调构件连接网络与令牌调度网络. 图 5 说明了该体系结构.

构件转换是基于在网络的端口之间流动的令牌进行调度，优先级高的令牌，将最先在端口之间流动，当触发条件满足时，构件转换被执行，即构件调度的优先级实际上是由输入输出端口的令牌的优先级来决定的. 构件框架的计算模型，既决定了在同一个端口上的多个令牌的优先级，也决定了整个构件连接网络上流动的所有令牌的优先级.

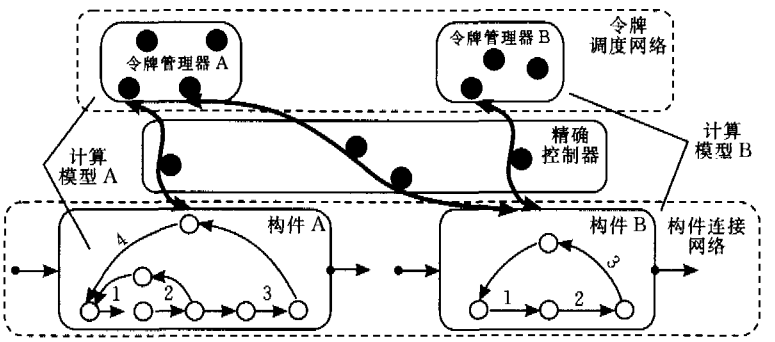


图 5　令牌调度网络

4.2　转换调度队列

PCC 维护两个全局队列：正在调度的转换队列 Q 和以时间排序的(等待调度的)转换队列 W. 只要 PCC 按一定的时间周期 T 检查 W 中的转换，将下一个周期内触发的构件转换加入调度对列 Q，假设在系统时刻 t 时，处于周期$[T_0, T_0+T)$内，当 G_τ 的输入/输出端口条件满足时，如果 $GT=\infty$，表明转换 τ 没有实时的调度要求，显然可以在下一个周期内执行，因此进入调度队列 Q；如果 $GT\in[t, T_0+2T)$，表明转换 τ 应在队列 Q 所调度的时间范围内触发，因此也进入 Q；否则转换 τ 进入 W. 若 $GT<t$，显然转换 τ 必须作另外的超时处理.

如图 6 所示，PCC 将调度队列 Q 分作三个级别：在当前时间进行实时计算的转换队列 Q_{realtime}、稍后某个时间进入执行阶段的转换队列 Q_{CT} 和 Q_{NT}、可在任意时间执行的转换队列 Q_{shared}. PCC 用事件控制器和任务控制器(具有各自的处理器上下文环境)分别执行 Q_{realtime} 和 Q_{shared}. Q_{realtime} 的来源是事件，其中部分转换是因时间触发的，部分是由异常处理例程抢占事件控制器后加入的. Q_{CT} 和 Q_{NT} 是介于实时队列 Q_{realtime} 和等待队列 W 之间的缓冲队列. Q_{shared} 的来源是任务，当 Q_{realtime} 为空时，任务控制器执行 Q_{shared}. Q_{CT} 和 Q_{NT} 中的转换具有明确的触发时间，分布于当前的时间周期 CT 和下一个时间周期 NT 内. Q_{CT} 将 CT 划分为 n 个时间片$\{CT_1, CT_2, \cdots, CT_n\}$，比如 CT 的跨度为 1s，$n=1000$，那么时间片的精度为 1ms，每个时间片内执行的所有转换串联为一条链表. 类似的，Q_{NT} 由 NT 内的 n 条转换链表组成. 假设当前时间所在的时间片为 CT_i $(1\leq i<n)$，则当事件控制器执行完 Q_{realtime} 中所有转换后，设置定时器在 CT_{i+1} 激活，定时器中断处理例程将把 Q_{CT} 队列中处于 CT_{i+1} 内的转换加入到 Q_{realtime}. 若 $i=n$，则 Q_{CT} 为空，事件控制器将 Q_{CT} 与 Q_{NT} 做对换，然后在 Q_{shared} 加入一个任务，该任务将 Q_{realtime} 队列中处于下一个周期内触发的转换加入调度对列 Q_{NT}. 事件控制器或任务控制器执行转换后，

都有可能改变构件连接网络上令牌流动的结构，满足某些触发器条件，从而引起 $Q_{realtime}$，Q_{shared}，Q_{CT}，Q_{NT} 队列的变化.

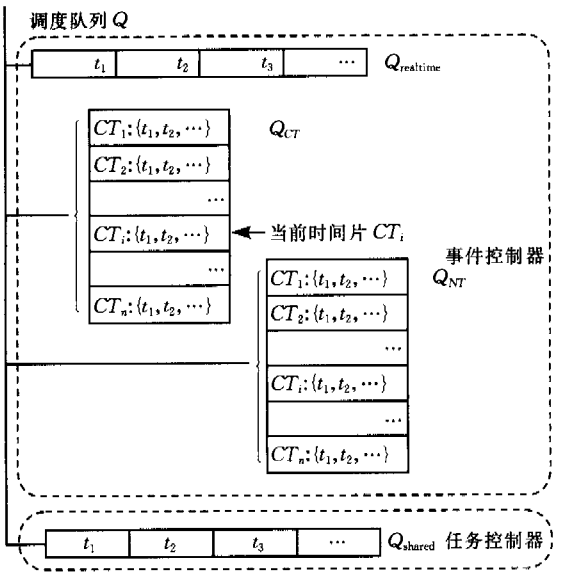


图 6 三级调度队列

4.3 事件控制器

$Q_{realtime}$ 由事件控制器执行，队列中的转换主要来源于硬件中断（包括定时器中断和其它硬件的中断信号）等异常处理，称为事件转换. 异常处理分解为两阶段，第一阶段是直接的中断例程，在屏蔽中断的情况下，执行一些关键计算（例如访问中断寄存器），然后将异常信号写入 PCC 的实时 FIFO. 实际上 FIFO 的作用相当于：如果一个构件转换要求读取端口的数据来自硬件中断等异常信号，那么异常处理以 FIFO 机制替代标准的 *WritePort* 函数实现，将异常信号转化为令牌，进入令牌调度网络，构件转换则通过标准 *ReadPort* 函数操作得到异常信号. 异常处理完成后，事件控制器得到处理器控制权，执行 $Q_{realtime}$.

假设 $Q_{realtime}=\{\{R_1,A_1,W_1\},\{R_2,A_2,W_2\},\cdots,\{R_n,A_n,W_n\}\}$，$\{R_i,A_i,W_i\}$ 表示队列中第 i 个转换的 *ReadPort* 函数、*Action* 函数和 *WritePort* 函数. 事件控制器对 $Q_{realtime}$ 的调度划分为三个阶段：$\{R_1,R_2,\cdots,R_n\}$，$\{A_1,A_2,\cdots,A_n\}$，$\{W_1,W_2,\cdots,W_n\}$. 事件控制器执行 $\{W_1,W_2,\cdots,W_n\}$ 使得构件行为所更改的状态同步到令牌通信网络中，这个同步阶段可能构造出新的事件并触发一些构件转换；同样的，FIFO 的异常信号也可能触发构件转换. 这些转换将按照实时要求分别加入 $Q_{realtime}$ 或 Q_{shared}，则事件控制器开始重新一轮循环；否则任务控制器得到处理器控制权，执行 Q_{shared}.

事件控制器的调度具有原子性，即多个同时到达当前时间的转换被重组为 $\{R_1,R_2,\cdots,R_n\}$，$\{A_1,A_2,\cdots,A_n\}$，$\{W_1,W_2,\cdots,W_n\}$，其中每个操作都是原子操作. 这一方面避免了高精度时间片切换带来的附加开销，另一方面保证一段指令执行的连续性，比如对一组有关联的硬件寄存器的连续读写或者访问构件内部状态的关键区.

4.4 任务控制器

Q_{shared} 由任务控制器执行，Q_{shared} 中的转换不具备时间触发条件，执行的计算行为与时间无关，称为任务转换. $Q_{realtime}$ 中的事件转换或者 Q_{shared} 中的任务转换可能改变框架状态，使得一些触发器被激活（比如端口接收到新的令牌）. 当事件控制器空闲时，任务控制器调度执行 Q_{shared} 中的转换，由于所有任务之间都是不可抢占的，只要端口接收到所需的令牌，意味着资源、数据等外部状态满足触发器条件，任务就可以一直执行，直到向端口输出令牌为止，因此令牌在端口间流动的先后顺序决定了任务的优先级高低.

任务控制器执行的 *Action* 函数，虽然与时间无关，但是计算结果却可能间接地与时间有关，比如对 $Q_{realtime}$ 队列进行调度，生成下一个周期 T 的 Q_{NT} 队列；PCC 在运行时动态执行新的任务，对相同触发时间（或者存在连续关系）的一系列转换 $\{\{R_1,A_1,W_1\},\{R_2,A_2,W_2\},\cdots,\{R_n,A_n,W_n\}\}$ 进行优化处理，合并或简化 *ReadPort*，*Action*，*WritePort* 三类函数，组合结果为 $\{R_{1,n},A_{1,n},W_{1,n}\}$，其中 $R_{1,n}=\{R_1,R_2,\cdots,R_n\}$，$A_{1,n}=\{A_1,A_2,\cdots,A_n\}$，$W_{1,n}=\{W_1,W_2,\cdots,W_n\}$，或者通过动态编译从构件转换的状态机生成二进制代码，提高事件控制器的性能，这是下一阶段的研究内容之一.

另外，为了使用第三方厂商提供的库、驱动等，任务控制器还可以分裂出多个单独的处理器上下文环境来支持二进制构件以多线程（或进程）形态运行，因此需要有一个调度器按照单独的时间片进行调度，这个调度器本身就是 PCC 中运行的一个构件，它具有周期性的事件，按照一定的调度算法来选择任务控制器中的上下文环境.

比较事件控制器和任务控制器，可以说事件控制器执行实时性计算，调度由事件（主要是时间和硬件中断）驱动的构件转换，强调系统响应的可靠性，一般来说，事件控制器所执行的 *Action* 函数是对事件的实时处理，适用于具有时间约束条件的计算模型；任务控制器执行与时间无关的计算，包含大量的

数据处理工作，控制器调度令牌在端口间的流动结构或者按顺序执行 *Action* 函数即可. 事件控制器和任务控制器所管理的转换队列之间不存在直接依赖关系，通过对全局的构件连接网络进行令牌调度实现协作.

5　实例研究

我们开发了一个远程协作系统，它运行在 PXA250 开发板上，也适于运行在 PDA 或手机平台上. 远程协作系统是基于网络的协作式数据的交互式共享平台. 系统中存在多种共享数据流，包括视频、音频、文字等各种数据格式，基于这些数据流，用户通过协作系统进行交流. 这种架构本质上是对多数据流的并发和实时处理.

视频对话是远程协作系统的重要部分，它既是嵌入式应用，同时也是典型的数据流应用. 它的软件结构包括如下五个部分：网络协议(VoIP)、视频采集、视频播放、用户界面、视频编码/解码. 前四个部分都是操作系统中运行的服务，并且与外围器件的驱动程序有关. 例如网络协议与无线网卡驱动有关，视频采集与摄像头驱动有关，视频播放既与视频加速硬件有关，也与显示驱动有关，用户界面与液晶屏/触摸屏/键盘驱动有关. 视频编码/解码软件通常是在 DSP 处理器上执行，因此操作系统需要提供 DSP 接口，例如 Linux 以设备文件的形式实现对 DSP 通信接口的驱动，因此视频编码/解码也是和驱动程序有关.

视频对话在传统操作系统上的运行形式如图 7 所示，程序与系统服务之间存在多个交互关系. 在视频对话期间，操作系统中的模块与应用程序有大量的数据通信. 如图中的虚线所示，视频信号经过摄像头驱动和视频采集后到达应用程序，在此分流为两种处理：(1)经过视频播放和显示驱动进行回放(即用户观察到自己的影像)；(2)经过视频编码、网络协议和网卡驱动后发送给对话方.

在视频对话过程中，许多进程进行频繁的通信，无论是采用 FIFO，PIPE 或者共享内存等机制，统称为进程间通信(IPC). 考虑最简单的软件架构，即每个模块只运行一个进程(或内核线程)，那么一个视频信号经过图 7 中虚线标示的路径，至少会引起 6 次 IPC 以及 7 次调度. 如果一个模块处理视频信号的时间较长，其间可能会发生更多次的(操作系统对多进程的)调度. 频繁的 IPC 和调度不仅影响了系统的性能，并且大大增加了功耗，在手机等移动终端中是比较严重的问题. 当采用主流的构件模型(如 COM，CORBA 等)开发时，上述这些模块都以构件的形式运行，由于构件间通信的实现机制是进程间通信＋桩代码，因此在视频数据流的处理路径中还需执行大量的桩代码，对执行的效率造成更严重的影响.

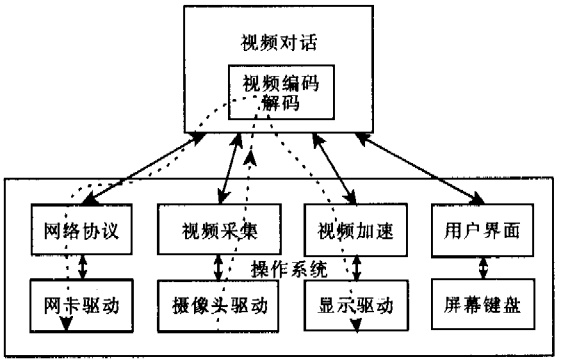


图 7　视频对话

相对地，在 Pcanel 操作系统的运行环境中，视频应用将视频数据流处理的计算模型直接运行在 PCC 上，因此 PCC 能够理解视频数据流在各个构件间流经的路径，直接实现令牌在构件之间的传递，消除大多数不必要的进程间通信和调度的开销，从而提高数据流应用的整体执行效率. 此外，PCC 通过事件控制器和任务控制器的协同工作，单独对构件的行为进行调度，避免了传统进程调度和同步的复杂性，因此能保证数据流处理的实时性.

我们分别在 Pcanel 和 Linux 上实现相同结构的视频对话应用，每个模块在 Pcanel 上实现为构件，在 Linux 上实现为线程，实验平台是 PXA250 开发板(核心频率 100MHz)，测试场景为同时运行 1～6 个视频对话应用(每秒 15 帧)，收集数据为每 1/15s 内处理器在特权模式(priviledged mode)的指令数，这些指令主要用于构件或进程的调度和通信. 图 8 所示的结果表明，Pcanel 的运行成本减少到 Linux 的三分之一以内，并发的应用越多，效果越显著.

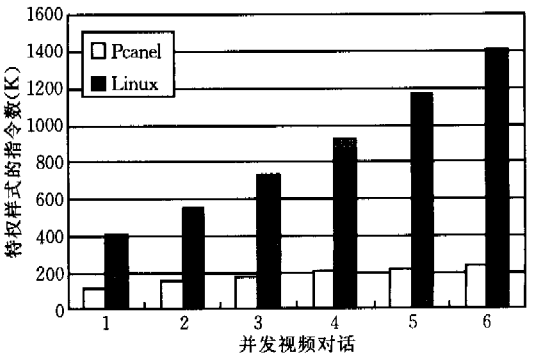


图 8　调度和通信的开销

6 相关工作

灵活的、可扩展的、构件化的操作系统内核已有诸多研究.大多数原型系统如 Pebble[4], 2K[5] 或者商业系统如 QNX, VxWorks 和 eCos,它们首先定义一组特别的、固定集合的核心接口,比如固定的任务或者线程模型、地址空间模型、中断处理模型或者通信模型.其余的扩展性的构件或模块虽然能够静态或者动态地链接到内核中,但是必须依赖这个固定的核心接口,因此并非完全遵从构件化方法的设计.eCos 支持构件的静态配置以及将构件打包成嵌入式操作系统,但是也同样依赖于一个预定义的基本内核,并且不提供动态重配置能力.OSKit[6] 仅仅是一系列用以构造操作系统的构件.总之,上述这些系统都缺乏清晰的计算模型支持嵌入式系统的数据流应用领域.

7 结束语

数据流应用在许多嵌入式系统中占据重要地位,为了面向这类应用领域实现构件化软件开发,本文提出了构件转换的精确控制核 PCC,它是构件化嵌入式操作系统 Pcanel 的构件模型的核心运行环境,通过计算模型对构件进行组合和调度.PCC 支持构件以令牌通信方式进行协作,形成构件连接网络.基于任务和事件的并行处理和实时响应,PCC 实现高性能的执行框架.PCC 采用精确构件转换模型,通过事件控制器和任务控制器的组合协作管理令牌调度队列,适于在嵌入式系统领域实现大量并发处理的数据流应用.经测试,Pcanel 有效减少了构件的调度和通信开销.下一阶段研究重点是将 PCC 扩展到多核处理器架构,研究分布式的令牌调度网络的性能以及对周期性任务的实时调度.

参 考 文 献

1 Xie Cheng, Chen Wen-Zhi, Shi Jiao-Ying. Pcanel: A model driven component framework. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Hague, Holand, 2004, 5171~5176

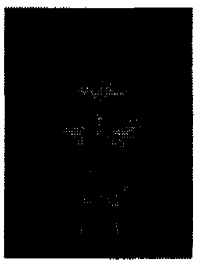
2 Karsai G., Sztipanovits J., Ledeczi A., Bapty T.. Model-integrated development of embedded software. Proceedings of the IEEE, 2003, 91(1): 145~164

3 Gamma E., Helm R., Johnson R., Vlissides J.. Design Patterns. Addison-Wesley, 1995

4 Gabber E., Small C., Bruno J., Brustoloni J., Silberschatz A.. The Pebble component-based operating system. In: Proceedings of the 1999 USENIX Annual Technical Conference, Monterey, CA, 1999, 267~282

5 Kon F., Campbell R. H., Mickunas M. D., Nahrstedt K., Ballesteros F. J.. 2K: A distributed operating system for dynamic heterogeneous environments. University of Illinois at Urbana-Champaign: Technical Report UIUCDCS-R-99-2132, 1999

6 Ford B., Back G., Benson G., Lepreau J., Lin A., Shivers O.. The Flux OSKit: A substrate for kernel and language research. In: Proceedings of the SOSP'97, Saint-Malo, France, 1997, 38~51



CHEN Wen-Zhi, born in 1969, Ph. D., associate professor. His current research interests include embedded systems & realtime system, distributed computing, media network system and technology, storage area network.

XIE Cheng, born in 1977, Ph. D. candidate. His research interests include embedded systems, computer architecture.

SHI Jiao-Ying, born in 1937, professor and Ph. D. supervisor. His research covers distributed and parallel computing, virtual reality, computer aided design and computer graphics.

Background

This work was supported by the National High Technology Research and Development Program (863 Program) of China under Component-based Embedded Operating System and Developing Environment (No. 2004AA1Z2050) and the Science and Technology Program of Zhejiang province under Novel Distributed and Real-time Embedded Software Platform (No. 2004C21059). The projects aim at developing component-based embedded operating systems based on component, middleware and real-time technologies, for configurability, reusablity, adaptibility, portablity and robustibility.

The paper describes the OS kernel, called Precise Control Core (PCC), which is based on a component transition model. PCC partitions states of component into quiescent states and executing states. The computation of concurrent components is recomposed into a sequence of split-phases of transitions. A wait-free synchronization technique is used preventing priority inversion. The collaboration of event controller and task controller of PCC achieves massively concurrency, thus supports using components in embedded systems.