第33卷 第16期 计 算 机 工 程 2007年8月
Vol.33 No.16 Computer Engineering August 2007
·软件技术与数据库· 文章编号：1000—3428(2007)16—0053—03 文献标识码：A 中图分类号：TP393

嵌入式实时操作系统 AutoOSEK 的设计

章亮飞，李银国
（重庆邮电大学汽车电子与嵌入式系统研究所，重庆 454000）

摘 要：OSEK/VDX 规范是一个用于汽车电子、并带有接口的开放式软件规范。基于 OSEK/VDX 规范，该文介绍了嵌入式实时操作系统 AutoOSEK 的内核结构，AutoOSEK 采用了与“硬件无关”、“硬件相关”部分完全独立的设计架构，讨论了这 2 个部分的实现方法，分析了系统性能。系统在基于 Motorola HCS12 系列芯片和 ARM 内核的多硬件平台中得以实现，并在汽车电子控制系统开发中得到了较好的应用。

关键词：嵌入式实时操作系统；OSEK/VDX 规范；中断管理

Design of Embedded Real-time Operating System AutoOSEK

ZHANG Liang-fei, LI Yin-guo
(Institute of Automotive Electronics and Embedded System, Chongqing University of Posts and Telecommunications, Chongqing 454000)

【Abstract】OSEK/VDX is a software standard with an open-ended architecture for distributed control units in vehicles. The structure of AutoOSEK which is an realization of embedded real-time operating system(RTOS) based on OSEK/VDX standard is introduced. The method of separated codes related with processor is adopted by AutoOSEK for satisfying reliability, both of the idea of realization of part irrelated with processor and part related with processor are discussed, and the performance of AutoOSEK is analyzed. AutoOSEK is realized on the chip of Motorola HCS12 and multi-hardware platform based on ARM kernel, and applied well to the automotive electronic control system.

【Key words】 embedded real-time operating system; OSEK/VDX standard; interrupt management

OSEK/VDX(简称OSEK)规范是欧洲汽车行业为满足各种软件间的兼容性和协作性而定义的一组带有通用服务接口的开放式软件规范[1]。AutoOSEK是遵循OSEK规范自主设计开发的嵌入式实时操作系统，完全实现了OSEK规范定义的所有应用程序接口。为了满足硬件平台的多样化，减少移植的复杂性，AutoOSEK采用了与“硬件无关”和“硬件相关”模块完全独立的架构。

1 内核结构

在 AutoOSEK 的设计中，将操作系统按功能分为：任务管理和调度，事件机制，资源管理，消息机制，警报和计数管理，中断服务程序和中断管理等模块，如图 1 所示。

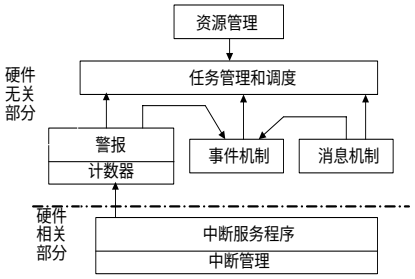

图 1 AutoOSEK 内核结构

任务管理和调度是核心部分，其他管理机制提供不同的服务支持。警报管理为任务提供了定时和循环处理机制；事件机制为扩展任务提供了同步机制；资源管理用来实现不同任务间互斥访问共享资源；消息机制实现任务间通信。中断服务程序和中断管理为以上管理机制的实现提供服务。图 1 中的箭头表示 AutoOSEK 中管理的实体间的关系。

2 硬件无关部分的设计

为方便系统在多硬件平台间的移植，将 AutoOSEK 中任务管理和调度、事件机制、资源管理、消息机制、警报和计数管理等模块，设计为与“硬件无关”的部分。

2.1 任务管理和调度

OSEK 规范将任务分为基本任务和扩展任务。基本任务具有 3 种状态：运行状态，就绪状态和阻塞状态。扩展任务多一个等待状态。

OSEK 规范定义了 4 种有效的符合类，分别为 BCC1,BCC2,ECC1,ECC2。一个“符合类”被定义为操作系统要求的一个具体实现，这些要求包括一个由应用程序指定的属性集[2]。其中，BCCx符合类只支持基本任务，ECCx符合类支持基本任务和扩展任务；xCC1 符合类要求同一优先级只有一任务，xCC2 符合类允许不同任务占用同一优先级。

xCC1 符合类要求同一优先级只有一任务，在调度时可按任务优先级调度，其实时性和可靠性较高，同时减少了复杂性。AutoOSEK 在实现时，采用的符合类级别为 ECC1，支持基本任务和扩展任务，满足了汽车电子控制系统设计的要求。

操作系统中的任务同步通过任务调度来实现。OSEK 规范支持 3 种调度策略：非抢占任务调度，全抢占任务调度和混合抢占任务调度。其中，混合抢占任务调度既可将一任务表示为非抢占型，也可表示为抢占型。AutoOSEK 完全实现

基金项目：国家“863”计划基金资助项目(2004AA1Z2380)；重庆市自然科学基金资助项目(CSTC 2005BB2071)
作者简介：章亮飞(1981－)，男，硕士研究生，主研方向：嵌入式系统；李银国，教授、博士
收稿日期：2006-10-11 **E-mail：**zhlfly@gmail.com

了这 3 种调度策略，并且通过共用内部资源方式定义任务组，使一组任务能同时具有抢占或非抢占性的特征，这一设计增强了不同应用选择任务调度策略的灵活性。

在 AutoOSEK 中每个任务都通过任务控制块(TCB)来管理，TCB 保存了该任务的主要信息和属性。

2.2 事件机制

事件机制是 OSEK 规范中的一种重要同步机制，只应用于扩展任务。事件是实现扩展任务在等待状态和就绪状态间转换的标志。AutoOSEK 中事件管理实体采用 TCB 中的双掩码结构，掩码的每一位对应一个事件，事件的具体含义由用户定义。

2.3 资源管理

在抢占任务调度方式中，AutoOSEK 为防止优先级的反转和死锁，采用了优先级天花板协议(PCP)，功能如下：

(1)资源的优先级下限大于或等于访问这个资源的所有任务中最大的优先级。

(2)资源的优先级上限小于所有不访问该资源，优先级大于在(1)中出现的最大优先级的任务的最小优先级。

(3)当任务对资源加锁时，任务的优先级暂时被设为该资源的优先级。

(4)当任务对资源解锁时，任务的优先级还原为它定义的静态值。

2.4 消息机制

任务之间是通过消息实现数据通信的，一条消息只能由一个任务发送，但可以由一个或多个任务接收。AutoOSEK 支持两类消息：(1)队列消息，内部数据结构被组织成 FIFO 队列，能够被接收服务程序移走；(2)非队列消息，可以被应用程序读取多次，每次都返回最后一次接收到的数据，即读操作不耗费数据。

2.5 警报和计数管理

OSEK 规范中没有定时器的概念，而是通过定义警报来实现定时器的功能，满足嵌入式控制系统根据所出现的一系列事件采取行动的需要。AutoOSEK 通过定义计数器对象来辅助实现该功能。当一个事件出现时，它就增量计数。当对应的计数器到达一个预定值时，警报就被触发，以实现应用程序需要完成的行为。

3 硬件相关部分的设计

AutoOSEK 为便于系统方便的移植到基于 8 位到 32 位微处理器的各硬件平台上，设计时充分考虑了使硬件相关部分代码最小化，并将其与硬件无关部分代码完全独立出来。

AutoOSEK 与硬件相关部分主要包括中断服务程序和中断管理。笔者参考了文献[3,4]，由于不同微处理器有不同的字长，因此不使用 C 的 short,int,long 等与编译器相关的数据类型，而是重新定义了一系列类型，以确保系统具有较强的移植性。

3.1 堆栈空间分配

AutoOSEK 为每个任务分配一个单独的堆栈区，每一堆栈区的空间大小可静态地独立配置。同时为了满足“占用较少空间”的要求，在设计时，优化堆栈空间的分配为

(1)系统中专设一中断堆栈区，以供中断服务程序执行期间使用。从而毋需在每一任务堆栈区中为中断服务程序预留堆栈存储空间。

(2)由于在非抢占任务调度模式下的基本任务或在混合抢占任务调度模式下，设定为“非抢占型”的基本任务间不存在同时使用堆栈的情况，这组任务共用一个堆栈区，空间大小设为该组任务中占用堆栈空间最多的任务占用堆栈空间的大小。

(3)同一任务组的任务由于具有“同时抢占”或“非抢占性”的特征，则同一任务组的基本任务间也不存在同时使用堆栈情况。这组基本任务共用一个堆栈区，堆栈区空间大小设置与(2)类似。

3.2 中断服务程序

操作系统中断服务程序包括：软中断服务程序 OSTaskForceDispatchHandler()，时钟节拍中断程序 OSISRSystemTimer()，所有中断服务程序为实现任务切换在退出前调用的通用服务 OSLeaveISR()。

(1)OSLeaveISR()

OSLeaveISR()服务是被中断服务程序在服务执行完成时，系统由中断态转向任务态时调用，以实现任务切换功能，代码如下：

```
void OSLeaveISR (void)
{OsRunning = OsHighPrioRdy; //得到就绪的最高优先级任务
OSTCBCur = OSTCBPrioTbl[OsRunning]; //取得该任务的 TCB
if( (OSTCBCur->OSTCBStat & OSTCBFIRST) != 0 )
{//如任务是第 1 次运行
OSTCBCur->OSTCBStat &= (OSBYTE)(~OSTCBFIRST);
OSLOADSP(OsTaskTOS[OsRunning]);
//将堆栈指针指向该任务堆栈栈顶位置
OSJUMPTASK(OsTaskEntry[OsRunning]);
//跳转到该任务入口处，执行任务
return;
}else{//如任务先前是被抢占或等待事件的任务
OSLOADSP(OSTCBCur->OSTCBStkPtr);
//得到该任务的堆栈指针；恢复处理器寄存器；执行中断返回指令}}
```

根据上述设计，系统通过 OSTCBFIRST 标志来区别任务是否为第 1 次运行，从而在恢复现场时进行不同处理。在新建任务时，毋需将该任务堆栈初始化成发生中断后保护现场的状态，使得系统毋需为每一具体硬件平台的实现独立开发初始化堆栈模块，减少了移植的复杂性。

(2)OSTaskForceDispatchHandler()

在 AutoOSEK 中，任务切换是通过软中断服务程序来实现的，代码如下：

```
void OSTaskForceDispatchHandler( void )
{保存处理器寄存器;
OSSAVESP(OSTCBCur->OSTCBStkPtr);
//在当前的 TCB 中保存当前任务的堆栈指针
OSLeaveISR();
```

(3)OSISRSystemTimer()

在 AutoOSEK 中，警报和计数管理是通过时钟节拍中断来实现的。AutoOSEK 支持中断嵌套，并专设一中断堆栈区，代码如下：

```
void OSISRSystemTimer ( void )
{保存处理器寄存器；
OsIntNesting += 1; //中断嵌套数加 1
if(OsIntNesting == 1)
{//系统由运行任务到中断服务程序
OSSAVESP(OSTCBCur->OSTCBStkPtr);
//在当前的 TCB 中保存当前任务的堆栈指针
OSLOADSP(OSISRTOS);
//将堆栈指针指向中断堆栈栈顶位置
}else{//抢占其它中断服务程序
```

```
OSAVESP(OSISRSP); //保存当前中断堆栈的指针}
   给产生中断的设备清中断；
   重新允许中断；
OsTimeTick();
//完成计数器计数，判别是否有警报到期，以进一步处理
OsIntNesting--;
if(OsIntNesting == 0)
{ //系统由运行中断服务程序到任务
if(OsHighPrioRdy > OsRunning)
{//就绪的最高优先级任务不为当前被中断运行的任务
OSLeaveISR ();
}else{//当前被中断运行的任务仍为就绪的最高优先级任务
OSLOADSP(OSTCBCur->OSTCBStkPtr);
//得到该任务的堆栈指针}
}else{//执行被抢占的中断服务程序
OSLOADSP (OSISRSP);
//得到即将重新运行的中断服务程序的堆栈指针}
恢复处理器寄存器；
return;}
```

3.3 中断管理

OSEK 规范定义了 2 类中断服务程序，即：

(1)中断服务程序中不能调用系统服务，中断执行时间最短，不能完成任务切换；

(2)中断服务程序中需调用系统服务，可完成较复杂功能的要求。AutoOSEK 提供了 2 对中断屏蔽服务完成中断管理，分别为：

1)DisableAllInterrupts()/EnableAllInterrupts()

2)SuspendOSInterrupts()/ResumeOSInterrupts()

当调用 DisableAllInterrupts()时，使所有中断无效，系统保存所有中断的当前状态，这些状态在 EnableAllInterrupts() 被调用时恢复，但不能应用第 1)对服务，使临界区嵌套。

第 2)对服务只针对第(2)类中断，对第(1)类中断不影响，并且第 2)对服务可采用嵌套调用。

4 性能分析

AutoOSEK 在基于 Motorola HCS12 系列微控制器开发的汽车 ABS 嵌入式系统的应用中，具有较高的可靠性、实时性。同时 AutoOSEK 可根据具体硬件环境进行配置和裁减，优化了堆栈管理，保证系统占用较少的 RAM 和 ROM，在实际应用中，系统可被裁减到 8KB 以下，从而满足在硬件条件较苛刻的环境下的实现。

由于系统在设计时充分考虑了支持不同硬件平台的实现，将硬件相关部分与无关部分代码完全独立，因此系统具有强移植性。

在系统生成时，可根据具体要求配置调度方式，定义包括任务、资源、事件、报警、消息和中断等各系统对象的属性和相互间的作用关系，使系统可配置性和扩展性增强。

5 小结

本文介绍了基于 OSEK 规范实现的嵌入式操作系统 AutoOSEK 的内核结构，根据与“硬件无关”和“硬件相关”2 个部分，分析了设计思想。由于在设计时，笔者充分考虑了系统的可移植性，使系统支持“基于从 8 位~32 位微处理器的多硬件平台上”的实现。实际应用验证了该系统的有效性。

参考文献

1 OSEK/VDX Group. OSEK/VDX Operating System[DB/OL]. (2004-07-05). http://www.OSEK-vdx.org.

2 Joseph L. OSEK/VDX 汽车电子嵌入式软件编程技术[M]. 罗克露, 译. 北京: 北京航空航天大学出版社, 2004.

3 Labrosse J J. 嵌入式实时操作系统μC/OS-II[M]. 邵贝贝, 译. 北京: 北京航空航天大学出版社, 2003.

4 Morton T D. 嵌入式微控制器[M]. 严隽永, 译. 北京: 机械工业出版社, 2005.

~~~~~~

(上接第 52 页)

集作为输入的运行情况。从图 10 中可以看出随着支持数的下降，MFCPLG 算法的运行时间差距不大，而 CARPENTER 的运行时间则快速增长，并且在支持数小于 4 后 MFCPLG 比 CARPENTER 要快 4 倍以上。

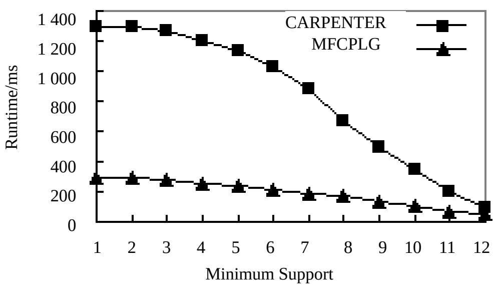

图 10 Mlung 数据集

4 问题讨论

由于在深度优先遍历行枚举树时，CARPENTER算法需要扫描整个$TT|_X$来生成$TT|_{X\cup\{i\}}$；而MFCPLG算法由$LG\text{-}tree|_X$构造$LG\text{-}tree|_{X\cup\{i\}}$的过程只需要遍历$LG\text{-}tree|_X$中以 i 为根结点的子树。所以，MFCPLG 算法是通过减少扫描时间提高了算法的时间性能。

虽然 LG-tree 以共享前缀的方式减小了采样条件所需的存储空间，但 LG-tree 中每个结点都要存储与其相应的模式，即基因集。所以随着基因的不断增加，MFCPLG 算法所需要的存储空间将迅速增加，算法的空间性能大打折扣。如何提高算法空间性能，是需要考虑解决的下一个问题。

参考文献

1 Pan F, Cong G, Tung A K H, et al. CARPENTER: Finding Closed Patterns in Long Biological Datasets[C]//Proc. of ACM SIGKDD Int’l Conf. on Knowledge Discovery and Data Mining. 2003: 637-642.

2 Han J, Pei J, Yin Y. Mining Frequent Patterns Without Candidate Generation[C]//Proc. of SIGMOD’00. 2000.

3 Han J, Wang J, Lu Y, et al. Mining Top-k Frequent Closed Patterns Without Minimum Support[C]//Proc. of ICDM’02. 2002: 211-218.

4 Wang J, Han J, Pei J. Closet+: Searching for the Best Strategies for Mining Frequent Closed Itemsets[C]//Proc. of ACM SIGKDD’03. 2003: 236-245.
~~~~~~