文章编号: 1001－9081(2003)05－0089－03

嵌入式操作系统封装层的设计与实现

何先波[1,2], 钟乐海[1], 芦东昕[2]
(1. 四川师范学院 计算机科学系, 四川 南充 637002; 2. 中兴通讯 成都研究所, 四川 成都 610064)

摘 要: 文中对嵌入式操作系统封装的必要性作了较详细地论述, 并以商用嵌入式操作系统 VxWorks 对通信类产品的封装为例对封装层的设计与实现进行了探讨。

关键词: 嵌入式操作系统; 操作系统封装层; 实时操作系统; 应用编程接口 (API)
中图分类号: TP311.11 文献标识码: A

Design and Implementation of Embedded Operating System Encapsulated Layer

HE Xiang-bo[1,2], ZHONG Le-hai[1], LU Dong-xin[2]
(1. *Department of Computer Science, Sichuan Teachers College, Nanchong Sichuan* 637002, *China*;
2. *Chengdu Institute of Zhongxing Telecommunication Equipment Co., Ltd., Chengdu Sichuan* 610064, *China*)

Abstract: In this thesis, the necessity of encapsulated layer in embedded operating system is disussed in detail. And its design and implementation are explored by encapsulating communication products on commercial embedded system, Vxworks.

Key words: embedded operating system; operating system encapsulated layer; real time operating system; application programming interface(API)

1 前言

所谓嵌入式操作系统的封装是指为了能缩短特定领域的嵌入式系统软件的开发周期, 增强嵌入式操作系统功能和实现嵌入式程序的可移植性、可维护性和代码继承性而对所使用的特定嵌入式操作系统的再"加工"。

嵌入式操作系统封装层的目的之一是将具体的底层实现细节进行屏蔽, 给上层具体应用软件编程者提供统一规范的调用接口。特别是对底层不同操作系统之间差异的屏蔽, 大大降低了应用软件开发人员的开发难度, 也使开发出的应用软件有较好的可移植性、可维护性和代码继承性。另外, 针对许多商用实时操作系统核心部分源码的不开放性, 封装层一般会对进程调度、内存分配等核心功能进行重新定义和优化, VOS 层位于嵌入式操作系统与应用程序之间, 参与对许多对象的管理, 使用户能跟踪到一些比较靠近内核的东西, 对原有操作系统的适用性和效率都有所提高, 而且可提供丰富的调试手段和实时监控功能。提高了关键应用的效率从而使得程序的可调试性增强, 有利于早期发现故障, 缩短项目的开发时间。由于嵌入式实时操作系统封装层的使用, 使得一个项目中很多基本功能都可以在这个封装层实现, 而这个封装层是一个经过测试的、可重用的模块, 它基本上是比较可靠的。一个软件的核心部分比较可靠, 可以大大增加该软件的稳定性。

2 嵌入式操作系统封装层设的设计与实现

下面以对通讯类嵌入式设备为背景来讨论嵌入操作系统封装层的设计方与实现案。其总体设计框架如图 1 所示(箭头表示调用关系)。

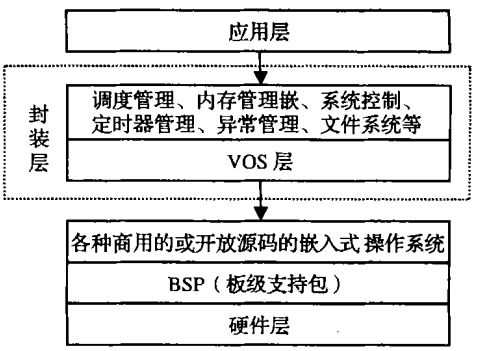

图 1 针对通讯类应用的嵌入式操作系统封装层结构框架

2.1 VOS(Virtual Operating System)层

与操作系统为了隔离具体的硬件环境而设计出 HAL 层(硬件抽象层)类似, VOS 层的功能就是为了隔离下层所使用的不同的操作系统, 从而给程序员提供一个与下层具体操作系统无关的统一的接口。封装层其他部分或应用程序只需调用 VOS 提供的标准功能接口, VOS 层识别该接口调用后, 直接或间接地路由到下层具体 OS 提供的相应接口调用。由于应用程序对具体的嵌入式操作系统的依赖减少, 从而便于上层应用软件在不同操作系统之间移植。VOS 层的设计主要考虑给上层应用提供一个什么样的与具体操作系统和硬件平台无关的调用接口集, 这是 VOS 实现的基础。该调用接口集必须建立在对所使用的各种操作系统提供的 API 以及对大量的应用软件调用需求进行分析的基础上, 同时要考虑到操作系统的发展趋势以及应用软件程序员的认同程度, 类 POSIX 接

收稿日期: 2002－11－21; 修订日期: 2003－01－13

作者简介: 何先波(1971－), 男, 四川苍溪人, 讲师, 硕士, 主要研究方向: 嵌入式操作系统; 钟乐海(1963－), 男, 四川广安人, 副教授, 博士研究生, 主要研究方向: 计算机网络、软件工程; 芦东昕(1971－), 男, 黑龙江哈尔滨人, 博士后, 主要研究方向: 计算机网络、嵌入式系统.

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

口规范应是一个比较好的参考。

一般而言，嵌入式操作系统得提供消息队列、调度控制、内存管理、时钟管理、I/O 驱动等基本接口，对这些接口功能的实现也可采取先实现一个最小的操作原语闭集，然后在实际应用中再做相应的扩充。考虑到 C 库是各种常见操作系统都提供了的，因此 VOS 层实现主要考虑标准 C 库以外的功能接口的实现，如进程或线程、信号量、计时器等。

下面以嵌入式操作系统 VxWorks 与通用操作系统 Windows NT 的互斥信号量为例利用组件和面向对象的思想通过虚函数机制考虑 VOS 实现方法，其它的实现与此类似。

该方法的基本思想是通过抽象类的成员函数提供统一的调用接口，通过对抽象类的继承对应到具体使用的操作系统，借鉴组件实现方法与虚函数功能最终完成路由。

VOS 层提供的互斥信号量调用就是 VOS-Mutex 类的成员函数：

```
class VOS-Mutex
狱
  public:
  virtual VOS-Mutex()狱狙;
  virtual void lock() = 0;
  virtual void free() = 0;
狙;
```

参照组件技术的实现方法，用一类厂类来提供 VOS 层实例化各接口类的方法：

```
class VOS-Factory
狱
  public:
  virtual Thread * createVOS-Thread(…) = 0;
  virtual VOS-Mutex * createVOS-Mutex() = 0;
  virtual VOS-Semaphore * createVOS-Semaphore(…) = 0;
  …
狙;
```

VOS-Thread，VOS-Mutex，VOS-Semaphore 等分别为线程，互斥信号量，同步信号量对应的 VOS 层接口抽象类。

通过对这些抽象类的继承而与各 OS 的实现相联系。

(1) 如果下层操作系统为 VxWorks，其对应的互斥信号量类 VxMutex 和类厂类 VxOSFactory 实现如下：

```
class VxMutex : public VOS-Mutex
狱
  private:
  SEM-ID hMutex;
  public:
  void lock()狱semTake(hMutex, WAIT-FOREVER);狙
  void free()狱semGive(hMutex);狙
  VxMutex()
  狱
    hMutex = semMCreate(SEM-Q-FIFO);
  狙
  VxMutex()狱emDelete(hMutex)狙
  void * getHandle()狱return (void *)hMutex;狙
狙;
```

semTake、semGive、emMCreate、semDelete 为 VxWorks 提供的产生互斥信号量的接口调用。

```
class VxOSFactory : public VOS-Factory
狱
  public:
  …
  virtual VOS-Mutex *createVOS-Mutex()
  狱
    return (VOS-Mutex *)new VxMutex();
                                    //对 VxWorks 的直接接口调用
  狙
  …
狙;
```

(2) 如果下层 OS 为 WindowsNT，相应的有关实现类描述如下：

```
class NTMutex : public VOS-Mutex
狱
  private:
  HANDLE hMutex;
  public:
  void lock()狱WaitForSingleObject(hMutex, INFINITE);狙
  void free()狱ReleaseMutex(hMutex);狙
  NTMutex()狱hMutex = CreateMutex(NULL, FALSE, NULL)狙
  NTMutex()狱CloseHandle(hMutex)狙
  HANDLE getHandle()狱return hMutex;狙
狙;
```

WaitForSingleObject、ReleaseMutex、CreateMutex、CloseHandle 皆为 NT 提供的系统调用。

```
class NTOSFactory : public VOS-Factory
狱
  public:
  …
  virtual VOS-Mutex *createVOS-Mutex()
  狱
    return (VOS-Mutex *)new NTMutex();
  狙
  …
狙;
```

有了这些实现后，可用一全局函数如 theOSFactory()来定位到系统当前使用的具体操作系统，该函数的说明如下：

```
VOS-Factory * theOSFactory()
```

该函数的功能就是用于实例化对应操作系统的类厂类。

如果当前使用的操作系统是 VxWorks，该函数的实现如下：

```
static VxOSFactory *VxOsF= NULL;
VOS-Factory * theOSFactory()
狱
  if (VxOsF == NULL)
    VxOsF = new VxOSFactory;
  return VxOsF;
狙;
```

如果当前使用的操作系统是 Windows NT 其实现则为：

```
static NTOSFactory *ntOsF = NULL;
VOS-Factory *theOSFactory()
狱
  if (ntOsF == NULL)
  狱
    ntOsF = new NTOSFactory;
  狙
  return ntOsF;
狙;
```

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

如果当前使用的操作系统为 VxWorks 只需用上述方法实现函数 theOSFactory 便可通过该函数进行相应的接口调用。如对要产生一互斥信号量调用路由如下:

VOS-Mutex mutex= theOSFactory()-> createVOS-Mutex()

当然也可采取宏以及类似 Java 虚拟机方法来实现,方法的选取主要应权衡实现上的难易程度,功能上的可扩充性以及时间代价性能等因素。

2.2　内存管理

在封装层中除了提供内存申请、分配与释放等功能外,还可考虑提供出错记录、统计查询,同时针对特定的应用领域采用更高效的分配算法等。从而弥补有些嵌入式操作系统在内存管理方面存在的如内存分配单元过大,内存浪费严重,要求内存物理连续,缺乏出错记录,缺乏内存安全机制等的不足。

从实现方法上看主要是利用下层操作系统提供的内存分配功能获取大块可用内存,然后对该块内存进行适当的处理后作为以后应用程序内存分配的基础。如可根据实际情况将该块内存进一步划分为不同大小的内存块,相同大小的内存块区可用一队列来进行管理。每小块内存可配一控制块来标识所属的队列、是否被占用、被占用的任务(进程/线程)等信息。该控制块还可考虑提供内存使用的统计信息,从而可以知道各种大小内存块的使用情况,有利于解决诸如内存泄漏之类的问题。

此外,内存安全问题也是许多嵌入式操作系统考虑较少的部分,封装层可给予适当的补充。如对内存访问越界问题的解决,一种方法可以对每块内存增加上、下边界标志,如果对一块内存的写操作越界,通常它的边界标志就会被改变,这样操作系统封装层就可以探测出来,并产生报警信息;另一种方法是如果下层操作系统提供了基本的 MMU 保护机制,则可考虑在内存块之间加"隔离带"的方法,如果访问越界,则产生页保护错。

2.3　调度管理

包括任务的建立、删除、调度、优先级调整、任务运行信息的查询与统计功能。由于有些嵌入式操作系统有优先级数的限制(如 VxWorks 只提供最多 256 种优先级),当应用需要启动的任务很多时有必要考虑子任务调度,实现二级调度,调度方式可基于消息的轮转。二级调度可以考虑避免对临界资源的互斥访问,这样可加快多个应用之间的消息通信,提高了系统的效率(特别是对提高呼叫处理能力有明显的帮助)。

2.4　定时器管理

提供基于消息接口的定时功能,可考虑支持相对定时、绝对定时、周期定时、延迟睡眠等功能。在定时器管理方面,不同的操作系统有很大的差别,且接口不同,同时支持的定时器个数也不一样。这就要求封装层的定时器管理提供接口统一、个数相对应用没有限制、功能丰富且更适合通信领域应用的定时功能。如果考虑到通讯类嵌入式系统有时使用的定时器最大可能达到上万个,这样如何合理地组织定时器队列,采用高效的访问算法从而最大限度地减少定时器队列的访问时间成为关键。

2.5　文件系统

在封装层中,最好考虑屏蔽底层文件系统的差别。底层操作系统的文件系统可以是 rt11(VxWorks 提供的文件系统),msdos 文件系统等,上层使用相同的访问接口。文件的存储介质相对上层应用也应是透明的,可以使用硬盘,也可以使用闪存(Flash),内存等。同时封装层的文件系统还可考虑提供嵌入式的交叉管理功能,前台的文件管理 Shell 可以与后台的文件管理器相连,支持操作维护台对前台文件系统进行的维护管理。另外很多嵌入嵌入式操作系统未提供文件访问的安全机制,封装层可根据具体的应用领域实现部分功能。具体的封装实现可参照 Linux 文件系统的实现方法。

2.6　系统监控

在封装层设计中系统监控主要考虑要完成对系统中与上层应用无关的系统软硬件资源的监控,如 CPU 使用情况,内存使用率,任务堆栈使用率,各个上层应用任务的运行次数,运行时间等等。同时还可考虑提供阈值告警,系列错误统计及上报等功能。该层的实现可考虑用一专门的任务(进程/线程)来定期地收集统计相关信息。

2.7　模块通信

模块通信可考虑提供面向上层应用的端到端的消息传送,接收者除了得到发送者发送的消息内容外,还可以得到消息的发送者、消息事件号、消息长度、消息类型等信息。模块通信不局限于单板之内,可以是机框内的多个单板之上的应用之间,甚至是多个机框内单板之上的应用任务之间的通信。这一切对上层应用来说是透明的,且接口相同,仅仅是参数不同。对单板之内的应用任务通信可考虑支持异步和同步两种方式,对单板之间支持异步通信。

2.8　异常处理

每种操作系统都有自己的异常处理功能,一般都采用异常处理钩子函数的方式与应用接口。操作系统的异常处理一般都是记录异常现场寄存器的值,这对事后分析很不直观。在封装层的异常处理中,可以考虑记录异常现场寄存器的值,发生异常的指令前后几十条指令,当前运行任务,任务当前处理消息的事件号等等,使得事后分析更直观一些。一般不采用记录文件的方式,因为异常发生的时候,文件系统不一定可用,就直接记录在 NVRAM、Flash 或特定位置的 RAM(复位而不断电,信息不会丢失)区中,且采用文本的方式记录,事后可以把记录信息传送到操作维护台上的文件中,供进一步分析用。

3　总结

当然,嵌入式操作系统封装层需要一定的时间和空间开销,会在一定程度上降低系统的效率,但通常这个开销都能控制在允许的范围内。同时在设计与实现时若考虑不周,也会引入一些故障,可通过严密测试和逐步完善使故障率降低到最小。

参考文献

[1]　汤子瀛,哲凤屏,汤小丹.计算机操作系统[M].西安:西安电子科技大学出版社,1996.
[2]　(美)Wolf W.嵌入式计算系统设计原理[M].孙玉芳,等译.北京:机械工业出版社,2002.
[3]　Tanenbaum AS, Woodhull AS.操作系统:设计与实现[M].王鹏,等译.北京:电子工业出版社,1998.
[4]　Labrosse JJ.嵌入式系统构件[M].袁勤勇,等译.北京:机械工业出版社,2002.
[5]　郑宗汉.实时系统软件基础[M].北京:清华大学出版社,2002.
[6]　杨万林.实时多任务软件开发技术支撑平台[R].深圳:中兴通讯技术中心,2002.

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net