**VxWorks内核解读-1**

2016年12月23日 20:10:06 [redseazhaojianertao](https://me.csdn.net/redseazhaojianertao" \t "_blank) 阅读数：2404

**前言：我曾经在做VxWorks培训期间跟身边的嵌入式工程师同行交流的时候，发现大家对嵌入式VxWorks系统的Wind内核不是特别了解，而网上对于VxWorks的Wind内核也没有系统性的解读与分析，因此我决定发表一系列的博文来系统性地解读Wind内核的设计思想。我选择的是VxWorks5.5系统的Wind 2.6版本内核（这个版本的代码网上都有分享O(∩\_∩)O），在分析的过程中，大家有任何的疑问，都可以给我留言，以便我进一步的完善博文，使得后来者阅读起来更为流畅。**

本文首先从实时内核的定义出发，对实时操作系统进行了介绍，并对实时操作系统的特点进行了说明，接着从内核的功能和结构角度介绍了整体式内核，层次式内核，以及微内核。最后对具有微内核特性的VxWorks Wind内核进行了介绍。

**1.1实时内核概述**

“实时”表示控制系统能够及时处理系统中发生的要求控制的外部事件。从事件发生到系统产生响应的反应时间称为延迟(Latency)。对于实时系统，一个最重要的条件就是延迟有确定的上界(这样的系统属于确定性系统)。满足这个条件后，根据这个上界大小再区分不同实时系统的性能。这里的“系统”是从系统论的观点讲的一个功能完整的设计，能够独立和外部世界交互、实现预期功能。这包括实时硬件系统设计、实时操作系统设计、实时多任务设计3部分。后两者可以概括为实时软件系统设计。实现实时系统是这3部分有机结合的结果。

从另外一个角度，即实时程度看，可以把系统分为硬实时系统和软实时系统。硬实时系统是这样一种系统，它的时间要求有一个确定的截止期限(Deadline)，超出截止期限的响应，即是计算无误，也是无法容忍的错误结果，通常会引起严重的后果，这样的系统属于硬实时系统。对于软实时系统来说，“实时性”仅仅是“程度”概念，在提交诸如中断、计时和调度的操作系统服务时，系统定义一个时间范围内的延迟。在该范围内，越早给出响应越有价值，只要不超出范围，晚点给出的结果价值下降，但可以容忍。

因此一个RTOS内核必须满足许多特定的实时环境所提出的基本要求，这些包括：

* 多任务：由于真实世界的事件的异步性，能够运行许多并发进程或任务是很重要的。多任务提供了一个较好的对真实世界的匹配，因为它允许对应于许多外部事件的多线程执行。系统内核分配CPU给这些任务来获得并发性。
* 抢占调度：真实世界的事件具有继承的优先级，在分配CPU的时候要注意到这些优先级。基于优先级的抢占调度，任务都被指定了优先级， 在能够执行的任务（没有被挂起或正在等待资源）中，优先级最高的任务被分配CPU资源。换句话说，当一个高优先级的任务变为可执行态(Wind内核中称为就绪态Ready State)，它会立即抢占当前正在运行的较低优先级的任务。
* 快速灵活的任务间的通信与同步：在一个实时系统中，可能有许多任务作为一个应用的一部分执行。系统必须提供这些任务间的快速且功能强大的通信机制。内核也要提供为了有效地共享不可抢占的资源或临界区所需的同步机制。
* 方便的任务与中断之间的通信：尽管真实世界的事件通常作为中断方式到来，但为了提供有效的排队、优先化和减少中断延时，我们通常希望在任务级处理相应的工作(uC/OS-III内核采用了这种策略)。所以需要任务级和中断级之间存在通信。
* 性能边界：一个实时内核必须提供最坏情况的性能优化，而非针对吞吐量的性能优化。我们更期望一个系统能够始终以50微妙(us)执行一个函数，而不期望系统平均以10微妙(us)执行该函数，但偶尔会以75微妙(us)执行它。
* 特殊考虑：由于对实时内核的要求的增加，必须考虑对内核支持不断增加的复杂功能的要求。这包括多进程处理(比如VxWorks RTP)，对更新的、功能更强的处理器结构(比如Multicore CPU)的支持。

**1．  实时硬件系统设计**

实时硬件系统设计是其它两部分的基础。实时硬件系统设计要求在满足软件系统充分高效的前提下，必须提供足够的处理能力。例如，硬件系统提供的中断处理逻辑能同时响应的外部事件数量、硬件反应时间、内存大小、处理器计算能力、总线能力等，以保证最坏情况下所有计算仍然得以完成。多处理的硬件系统还包括内部通信速率设计。当硬件系统不能保证达到实时要求时，可以确信整个系统不是实时的。目前，各种硬件速度不断提高，先进技术大量涌现，硬件在大多数应用中已经不是实时系统的瓶颈。因而，实时系统的关键集中在实时软件系统设计，这方面也成了实时性研究的主要内容，也是最复杂的部分。许多场合甚至对实时硬件系统和实时操作系统不加区分。

**1．  实时操作系统设计**

先来看实时操作系统性能评价的几个主要指标：

* 中断延迟时间：对一个实时操作系统来说，最重要的指标就是中断关了多长时间，所有实时系统在进入临界区代码段之前都要关中断，执行完临界代码之后再开中断。关中断的时间越长，中断延迟就越长。因此中断延迟时间可以表述为关中断的最长时间与开始执行中断服务子程序第一条指令的时间之和，有时也表述为从系统接收中断信号到操作系统做出响应，进入中断服务程序的时间。
* 中断响应时间：中断响应时间定义为从中断发生到开始执行用户中断服务子程序代码来处理这个中断的时间。中断响应时间包括开始处理这个中断前的全部开销。中断响应时间包含了中断延迟时间，因此在考虑一个实时系统对外部中断的处理时间时，通常指考虑中断响应时间。典型地将执行用户代码之前保护现场，将CPU 的各寄存器推入堆栈的时间记为中断响应时间。
* 任务切换时间：多任务之间进行切换所花费的时间，即从前一个任务开始保存上下文的第一条指令开始，到后一个任务恢复上下文开始运行第一条指令为止的时间段。

从实时性角度看，操作系统经历了前后台系统、分时操作系统和实时操作系统3个阶段。

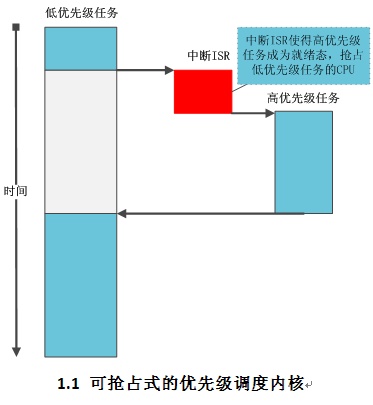
前后台系统(Foreground/Background System)其实没有操作系统，系统中只运行一个无限主循环，没有多任务的概念，但是通过中断服务程序响应外部事件。在前后台系统中，对外部事件的实时响应特性从两方面看。

* 中断延迟时间：主循环一般保持中断开放状态，因此前后台系统中断响应非常快，并且通常允许嵌套；
* 中断响应时间：需要经历一次主循环才能对中断服务程序中采集的外部请求进行处理，因此系统响应时间决定于主循环周期。

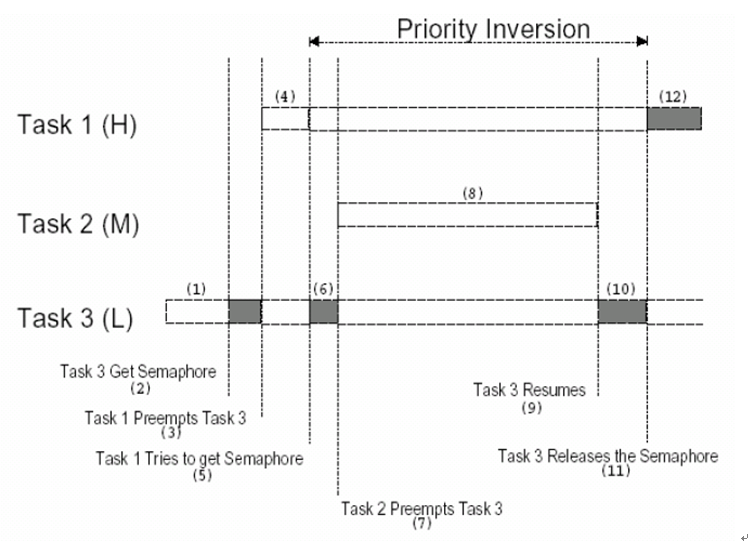
分时操作系统(Time-sharing Operating System)将系统计算能力分成时间片，按照一定的策略分配给各个任务，通常在分配过程中追求某种意义上的公平，分时操作系统不保证实时性。

实时操作系统(RTOS)的目的是实现对外部事件的实时响应，即根据前面对实时性的定义，实时操作系统必须在确定的时间内给出响应。实时操作系统必须满足下面4个条件：

* 可抢占式的优先级调度内核，当一个运行着的任务使一个比它优先级高的任务进入了就绪态，当前任务的CPU 使用权就被剥夺了，或者说被挂起了，那个高优先级的任务立刻得到了CPU的控制权，即始终保证优先级最高的任务拥有CPU使用权，如图1.1所示



* 中断具有优先级，且中断可嵌套，即高优先级的中断可以打断正在执行的低优先级中断处理程序，优先得到执行，等到其处理完毕再回到低优先级的任务继续执行，
* 防止优先级反转，系统服务的优先级由请求该服务的任务的优先级确定，具有优先级保护机制，以防止优先级翻转；优先级翻转式指当一个高优先级的任务被迫等待一个低优先级的任务不确定的完成时间的时候，优先级反转将会发生。考虑下面的情景：任务T1、T2、T3是三个优先级依次降低的任务。T3已经通过获取与保护资源相关的信号量，从而获取了这些资源。当T1抢占T3通过获取相同的信号量来竞争这些资源的时候，T3将会被阻塞。如果我们能够保证T1阻塞的时间不会长于T3正在执行后释放这些资源的时间，情况不会发生特别严重，毕竟T1占用的这个资源是不可抢占的。但是低优先级的任务是脆弱的，通常会被中等优先级的任务所抢占。例如此时的T2，这将会使T3无法释放T1需要的资源。这种情况将会一直存在，将会使得处于最高优先级的T1阻塞的时间不确定。示意图如1.2。



1.2优先级反转示意图

上述问题的一种解决方法是使用优先级继承协议(Priority Inheritance Protocol)。当高优先级任务需要低优先级任务占用资源时，将低优先级任务的优先级别提高到和高优先级同样的级别，即相当于低优先级任务继承高优先级任务的优先级级别；防止优先级翻转的另外一种协议是优先级天花板（Priority Ceiling），其设计策略是对优先级翻转采取“预防”，而不是“补救”。也就是说：不论是否阻塞了高优先级任务，持有该协议的任务在执行期间都被赋予优先级天花板所占用的优先级，以使任务尽快完成操作，因此可以把任务优先级天花板看作是更“积极”的一种保护优先级的方式，我们在后面的博文中将会详细分析者两种方案的设计与实现。

* 实时操作系统(RTOS)的性能评价指标(中断响应时间和任务切换时间)具有固定上界。

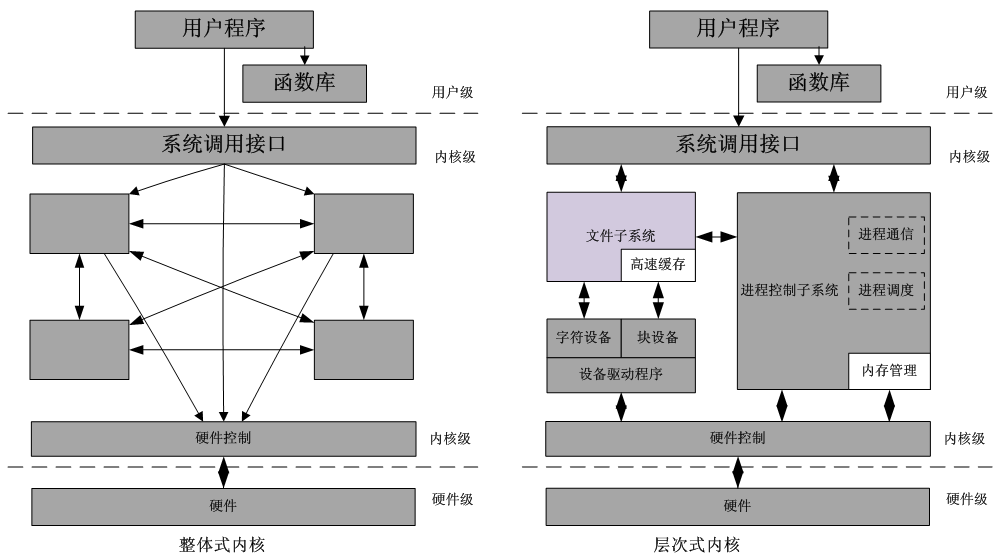
满足上面4个必要条件后，RTOS内核具体的实现机制就决定了其实时性的优劣。VxWorks的Wind内核是一个真正的实时微内核，满足上述条件。同时wind内核采取单一实时地址空间，任务切换开销非常低，相当于在UNIX这样的主机上切换到相同进程内的另一个线程，并且没有系统调用开销。高效的实时设计使Wind内核在从工业现场控制到国防、航空等众多领域中表现出优秀的实时性(严格的说用在航空领域的是VxWorks 653版本)。

**1.2微内核操作系统设计理念**

传统上，一个操作系统分为核心态和用户态。内核在核心态运行，为用户态的应用程序提供服务。内核是操作系统的灵魂和中心，决定了操作系统的效率和应用领域。在设计操作系统时，内核包含哪些功能以及内核功能采取何种组织结构，都是由设计者决定的。从内核功能和结构特点看，操作系统具有单体式内核、层次式内核、微内核三种不同形式。

单体式内核以过程集合的方式编写，链接成一个大型可执行的二进制程序。使用这种技术，系统中的每一个过程可以自由调动其它过程，只有后者提供了前者需要的一些有用的计算工作。这些可以不受限制，彼此调用的成千个过程，常常导致出现一个笨拙且难以理解的系统。单体式内核结构的操作系统不进行任何的数据封装和隐藏，在具有较高效率的同时，存在着难以扩展和升级的缺点。

层次式内核结构的操作系统将模块功能划分为不同层次，下层模块封装内部细节，上层模块调用下层模块提供的接口。Unix，Linux，Multics等属于层次结构操作系统。层次化使操作系统结构简单，易于调试和扩展。单体式内核和层次式内核结构如图1.3所示。



1.3整体式内核和层次式内核结构图

不管单体式结构，还是层次式结构，它们的操作系统都包括了许多将其用于各种可能领域时需要的功能，故被称为宏内核(Monolithic kernel)操作系统，整个核心程序都是以核心空间(Kernel Space)的身份及监管者(也称特权)模式(Supervisor Mode)来运行，以至于可以认为该内核本身便是一个完整的操作系统。以UNIX为例，其内核包括了进程管理、文件系统、设备管理、网络通信等功能，用户层仅仅提供一个操作系统外壳和一些实用工具程序。

用层次式方法设计操作系统，设计者需要确定在哪里划分内核-用户的边界。在传统上，所有的层都在内核中，但是这样做是没有必要的。事实上，尽可能减少内核态功能的做法更好，因为内核中的错误会快速拖累系统；相反可以把用户级任务设置位较小的权限，这样某一个错误的后果将不是致命的。微内核的设计思想是，为了实现其高可靠性，将操作系统划分为小的、定义良好的模块，只有其中一个模块(微内核模块)运行在内核态上，其余的模块由于功能相对较弱，则作为普通用户任务运行。特别的是地，由于把每个设备驱动和文件系统分别作为普通任务，这些模块中的错误虽然会使得这些模块崩溃，但不会使得整个系统死机。例如在音频驱动中的错误会使得声音断续或者停止，但是不会使这个计算机系统崩溃。相反在宏内核操作系统中，由于所有的设备驱动都在内核中，一个有故障的音频驱动很容易导致无效的地址引用，进而造成恼人的系统停机。

有许多微内核已经实现并投入应用。微内核在实时、工业、航空以及军事应用中特别流行，因为这些领域都是关键任务，需要有高度的可靠性。因此嵌入式操作系统大多采用微内核结构。微内核操作系统是近二十年新发展起来的技术，内核非常小但效率高，从数十KB到数百KB字节，适合于资源相对有限的嵌入式应用。微内核将很多通用操作的功能从内核中分离出来（如文件系统，设备驱动，网络协议栈等），只保留最基本的内容。知名的内核有Green Hills Software公司开发的INTEGRITY实时操作系统，windriver系统公司开发的vxWorks，黑莓手机(BlackBerry)制造商RIM(Research In Motion Ltd.，RIM)使用的QNX实时系统，以及L4、PikeOS、Minix3等。

备注：从微内核模块运行在CPU核心态上，其它模块运行在非核心态，这个角度来说，VxWorks的Wind内核并不能算是严格意义上的微内核系统。Wind内核通过全局变量kernelState模拟了Wind内核的特权态。Wind内核中以wind\*开头的例程构成Wind内核的核心服务例程。当kernelState为FALSE时，意味着Wind内核此时没有程序访问，VxWorks的外围模块可以调试Wind内核的核心服务例程；当kernelState为TRUE时，意味着其它程序正在使用内核态例程，当前需要调用核心服务例程的程序必须放置到延迟队列中，直到处于核心态的程序退出内核态(即kernelState为FALSE)，延迟的内核态例程才会被执行，从这里我们可以看出，kernelState模拟的核心态是非抢占式的。VxWorks的wind内核采用全局变量模拟了核心态服务例程，在加上其高度的可配置型，也具有一般微内核操作系统所具有的的特性。

一般认为微内核操作系统具有如下优点：

* 统一的接口，在用户态和核心态之间无需进行识别；
* 可伸缩性好，易于扩充，能适应硬件更新和应用变化；
* 可移植性好，操作系统要移植到不同的硬件平台上，只需修改微内核中极少代码即可；
* 实时性好，内核响应速度快，可以方便地支持实时处理；
* 安全可靠性高，微内核将安全性作为系统内部特性来进行设计，对外仅使用少量应用编程接口。

由于操作系统核心常驻内存，而微内核结构精简了操作系统的核心功能，内核规模比较小，一些功能都移到了外存上，所以微内核结构十分适合嵌入式的专用系统，如图1-4所示的VxWorks系统结构，大家可以直观的感受到Wind内核在VxWorks系统中的地位O(∩\_∩)O。

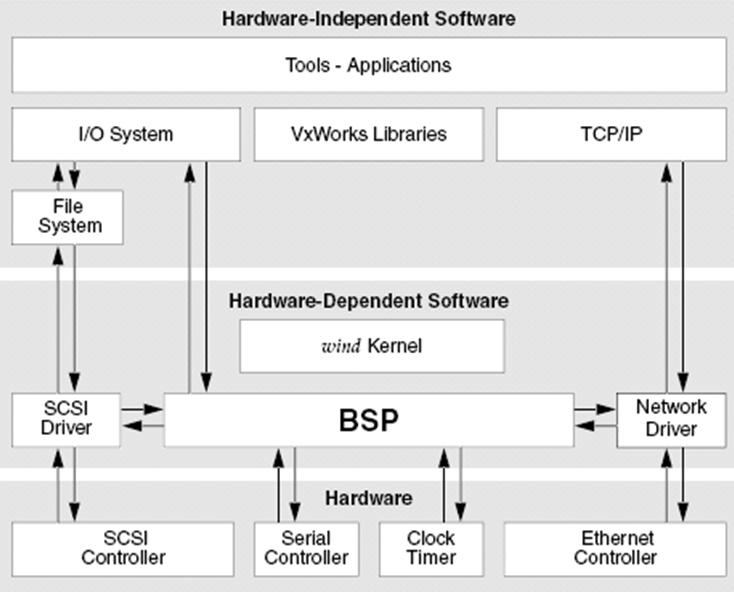


图1-4 VxWorks系统结构

**1.3 wind微内核设计**

为了提高微内核效率，有两种实现模式：受保护的虚地址空间模式和无保护的单一实地址空间模式。前者在宏内核操作系统（如Unix）和某些微内核操作系统（如QNX，Minix3）中采用。这种模式的优点是显而易见的：任务独立运行、不受其他任务错误影响、系统可靠性高。

VxWorks的Wind内核采取单一实地址空间模式，所有任务在同一地址空间运行，不区分核心态和用户态。其优势在于：

* 任务切换时不需要进行虚拟地址空间切换；
* 任务间可以直接共享变量，不需要通过内核在不同的地址空间复制数据；
* 系统调用时不需要在核心态和用户态之间切换，相当于直接的函数调用。

备注：系统调用需要从用户态切换到核心态，以执行用户态下不能执行的操作，在许多处理器上这是通过一条等价于系统调用的TRAP指令实现的，在执行该指令前要经过严格的参数检查。VxWorks中不存在这样的切换，因此系统调用和一般函数调用没有什么差别。但本系列博文中仍然沿用一般说法。

对于两种模式孰优孰劣，各自的支持者们进行了大量的争论。比较各有所长的东西往往非常困难。本文倾向于认为，对于嵌入式实时应用，单一实地址模式要合适一些。许多实践也证明，依靠虚地址保护来提高可靠性总存在局限性，毕竟程序运行出了错误，从某种程度上说虚地址保护只是使已经出现的错误经过一个延迟、积累和放大的过程，用过Windows就会有这种感触。而经过大量关键应用检验的VxWorks操作系统，则被充分证明是高度可靠的(当然了，可靠的系统必须由可靠的操作系统和可靠的应用系统组成)。

层次结构的Wind内核仅提供多任务环境、进程间通信和同步功能的服务例程。这些服务例程足够支持VxWorks在较高层次所提供的丰富性能的要求。VxWorks的Wind内核操作对于用户是不可见的。应用程序为了实现需要内核参与的任务管理和同步使用一些系统调用，但这些调用的处理对于调用任务是不可见的。应用程序仅链接恰当的VxWorks例程（通常使用VxWorks的动态链接功能），就象调用子程序一样发出系统调用。这种接口不象Linux内核需要一个跳转表接口，用户需要通过一个整数来指定一个内核功能调用。

**1.4 Wind内核类设计思想**

VxWorks采用类和对象的思想将Wind内核分成5个组成部分：任务管理模块、内存管理模块、消息队列管理模块、信号量管理模块、以及看门狗管理模块。

备注：除了上面的五个模块之后，还有虚拟内存管理接口(VxVMI)模块和TTY环形缓冲区管理模块也是Wind内核用类和对象思想管理的两个模块。虚拟内存接口模块(VxVMI)是VxWorks的一个功能模块，它利用用户片上或板上的内存管理单元(MMU)，为用户提供了对内存的高级管理功能；TTY环形缓冲区管理模块是ttyDrv设备的核心，ttyDrv设备称为虚拟设备，处在I/O系统和真正驱动程序之间形成了一个转换层，为VxWorks提供了一个标准的I/O接口，一个虚拟ttyDrv设备可以管理多个串行设备驱动程序。这两个模块严格意义上来说并不是一个RTOS内核理论上的组成部分，因此才不把它们列在Wind内核的组成部分上。

在wind内核中，所有的对象都是类的组成部分，类定义了操作对象的方法(Method)，同时维护着对所有对象的操作记录。Wind内核采用了c++的语义，但是采用c语言来实现。整个Wind内核通过显式编码实现，其编译过程并不依赖于具体的编译器。这意味着Wind内核不但可以在vxWorks自带的diab编译上编译，也可以使用开源的gnu/gcc编译器。VxWorks为Wind内核设计了一个元类(Meta-class)，所有的对象类(Obj-class)都是基于该元类。每个对象类只负责维护各自对象(Object)的操作方法(比如创建对象、初始化对象、注销对象等)、以及管理统计记录(比如一个创建对象的数据、销毁对象的数目等)。图1.5实现了元类、对象类、以及对象之间的逻辑关系。

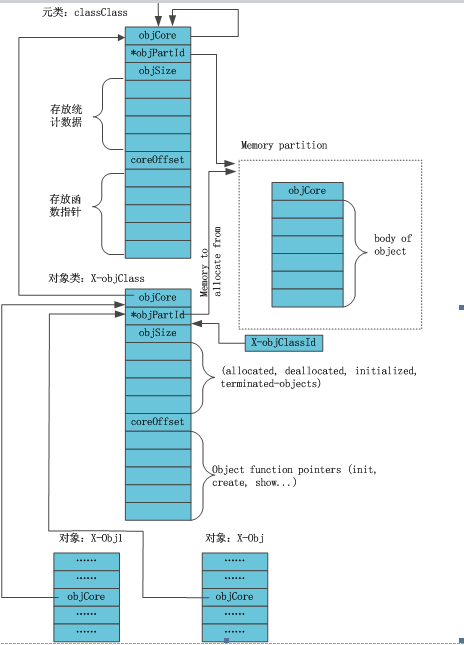


图1-5 元类、对象类、对象间关系图

备注：每次画这种类和对象关系图的时候，我就异常的纠结。因为在wind内核的设计中，元类classClass有一个ID号classClassId，每个对象类X-objClass也有一个ID号X-objClassId。在C语言的实现中classClassId和X-objClassId都是指针变量，存放的是相应类的地址。在初始化对象类和对象时，对象类和对象的objCore域存放的该指针变量的值(即相应类的地址)，跟classClassId没有太大的关系。图1.5真实地反应了objCore存放的类地址这个真实的关系。

如果从逻辑上来看的话，图1.6看起来更舒服，虽然从C语言实现来说，classClassId和X-objClassId的作用是错误的，但是从逻辑上看图1.6能更清晰的描述问题(并且图也更美观O(∩\_∩)O)，虽然图1.6上objCore存放的指向类地址的指针。正因为如此，在画类和对象关系图时，我采用图1-6所示的方式。

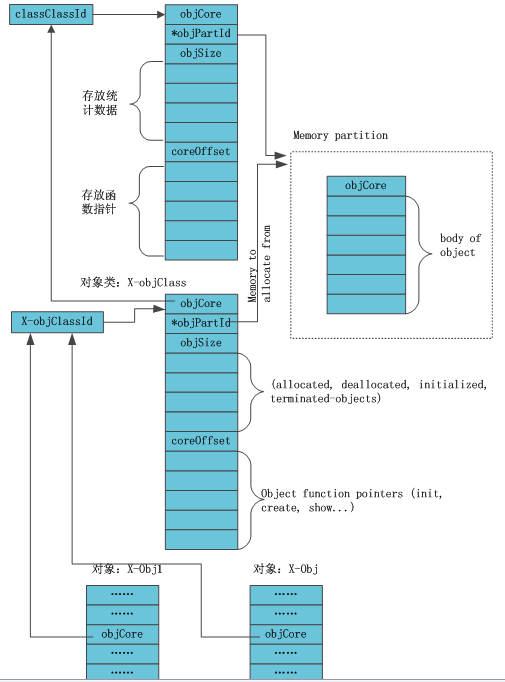


图1-6 元类、对象类、对象间关系图

正如图1-5，图1-6所示的那样每个对象类都指向元类classClass，每个对象类只负责管理各自的对象。Wind内核完整的元类、对象类、以及对象间逻辑关键，见图1-7。

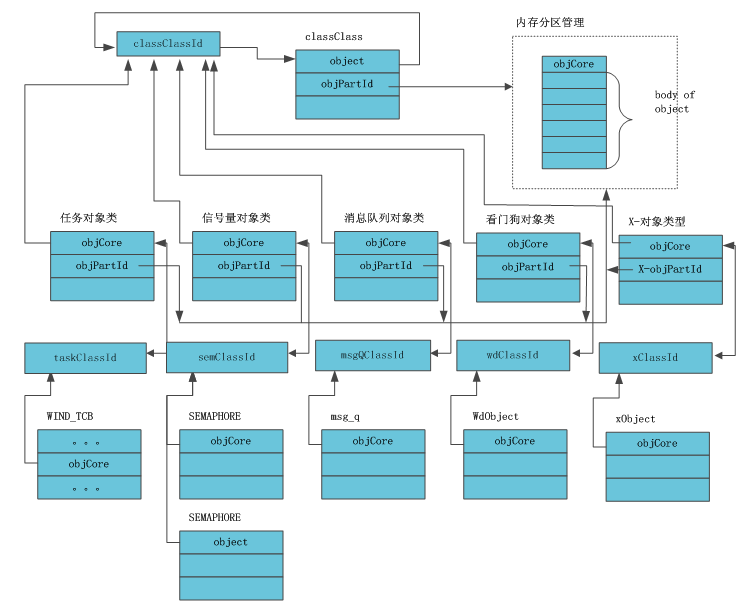


图1-7 wind内核各个组成模块间对象类、对象和元类的关系

备注：类管理模式不是Wind内核的特性，从功能上来说它仅仅是Wind内核组织各个模块的手段，所有内核模块的对象类、类对象都依赖于它。VxWorks采用类及对象来组织操作系统的结构，一个最重要的优势是增加了代码的安全性，即在创建新的对象类和对象、以及删除对象类和对象都可以对对象类、对象进行验证。

**1.5 Wind内核的特性**

VxWorks的Wind内核自然具有1.2节所描述的所有RTOS所共有的四个特性，其所有特点可以概括如下：

* 可裁剪的微内核设计；
* 多任务并发执行；
* 可抢占的优先级调度算法；
* 可选的时间片轮转算法；
* 任务间通信和同步机制；
* 快速的上下文切换时间；
* 低中断延时；
* 快速的中断响应时间；
* 可嵌套中断支持；
* 256个任务优先级；
* 任务删除保护；
* 优先级继承；
* 基于函数的调用，不采用陷阱指令和跳转表；
* VxWorks内核运行在处理器特权模式；
* VxWorks内核分层实现，VxWorks的核心库Wind内核处于核心态，由全局管理kernelState进行保护。

备注：本系列博文接下来的部分将详细分析wind内核如何进行设计，以具有这些特性。

再废话几句O(∩\_∩)O：我在分析Wind内核时所秉持的宗旨是策略(Mechanism)和机制(Policy)相分离的原则，策略(Mechanism)和机制(Policy)相分离是微内核设计的指导思想，换句话说微内核操作系统设计的指导原则是提供机制而不是策略。为了更清楚地说明这一点，我们以任务调度为例。一个简单的调度算法是为每一个任务赋予一个优先级，并让内核执行具有最高优先级的就绪任务。在这个例子中，机制(Mechanism)是在内核中寻找最高优先级的就绪任务并运行之；而策略(Policy)则是赋予任务相应的优先级。换句话说机制负责提供什么样的功能，策略则负责如何使用这些功能。策略和机制相分离指导思想可以使操作系统内核变得更小，更稳定。正如一句话说的好“一个优美的内核不是还有什么样的功能还可以增加，而是还有什么样的功能还可以减少”(哥们忘记是谁说的了⊙﹏⊙b汗)。

本系列博文力求在分析研究Wind内核的同时，思考RTOS的内核设计思想源泉。VxWorks的Wind内核经历了近20年的发展完善，达到目前的稳定状态。采用目前的这种设计、一定有其内部的考量，我希望尽可能的从一个系统设计者的角度来分析Wind内核的设计思想、工作机制、以及具有的特性，为我们设计一个款优秀的RTOS内核提供借鉴！

待续......O(∩\_∩)O哈哈~