VxWorks高级程序设计

李方敏 编著

清华大学出版社

VxWorks 高级程序设计

李方敏　编著

清华大学出版社

北 京

内 容 简 介

全书深入而系统地讲解了 VxWorks 高级程序设计的重点和难点，尤其对 POSIX 编程、I/O 系统、网络应用编程等作了详细的介绍，并给出了众多的实用编程技巧。同时，本书对于 VxWorks 中出现的新技术及其优秀特性也作了详细的介绍。

本书共 12 章，内容包括 wind 内核、任务间通信、POSIX 编程、信号、I/O 系统、文件系统、VxWorks 网络整体分析、网络应用编程、网络驱动（END）、BSP 概述、VxWorks 映像、VxWorks 启动过程等知识。本书内容详实、实例丰富、可读性强，是 VxWorks 中、高级开发人员的一本不可多得的参考书籍。

版权所有，翻印必究。

本书封面贴有清华大学出版社激光防伪标签，无标签者不得销售。

图书在版编目（CIP）数据

VxWorks 高级程序设计/李方敏编著. -北京：清华大学出版社，2004

ISBN 7-302-08127-1

Ⅰ. V… Ⅱ. 李… Ⅲ. 实时操作系统，VxWorks Ⅳ. TP316.2

中国版本图书馆 CIP 数据核字（2004）第 012635 号

出 版 者：清华大学出版社　　地　　址：北京清华大学学研大厦

http://www.tup.com.cn　　邮　　编：100084

社 总 机：010-62770175　　客户服务：010-62776969

责任编辑：曾　刚

封面设计：秦　铭

版式设计：郑轶文

印 刷 者：北京市清华园胶印厂

装 订 者：三河市化甲屯小学装订二厂

发 行 者：新华书店总店北京发行所

开　　本：185×260　印张：25.5　字数：570 千字

版　　次：2004 年 5 月第 1 版 2004 年 5 月第 1 次印刷

书　　号：7-302-08127-1/TP・5871

印　　数：1～4000

定　　价：36.00 元

本书如存在文字不清、漏印以及缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：(010)62770175-3103 或(010)62795704

目　录

此为试读，需要完整PDF请访问：www.ertongbook.com

第1章 wind 内核

1.1 内核概述

VxWorks 操作系统内核称为 wind 内核，下面从实时性能、核结构、调度特点等方面初步探讨。

1.1.1 实时内核

“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，“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，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底线（Deadline），超出底线的响应属于错误的结果，系统将会崩溃，上面所说的实时系统属于硬实时系统。对于软实时系统来说，“实时性”是个程度概念，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。

1. 实时硬件系统设计

实时硬件系统设计是其他两部分的基础。实时硬件系统设计要求满足在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系统和实时操作系统不加区分。

2. 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：从接收中断信号操作系统做出响应，并完成进入中断服务程序的时间；
- 任务切换时间：多任务之间进行切换所花费的时间；
- 系统响应时间：系统在发出处理要求到系统给出应答信号的时间。系统响应时间从整体上评价操作系统，综合了前面两个指标。

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。（1）中断延迟：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；（2）系统响应时间：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（Real Time OS，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几个条件：

- 可抢占的内核；
- 可抢占的优先级调度；
- 中断优先级；
- 中断可嵌套；
- 系统服务的优先级由请求该服务的任务的优先级确定；
- 优先级保护（优先级翻转保护）；
- 前述实时操作系统性能评价指标具有固定上界。

满足上面的必要条件后，内核内部具体的实现机制就决定了其实时性的优劣。**VxWorks** 的 wind 是一个真正的实时微内核，满足上述条件。同时，**wind** 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 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。

3. 实时多任务设计

具备前面两个实时条件后，实时系统的最终实现就取决于实时多任务设计。这部分是最能体现系统设计者艺术的部分，也是最有挑战性的部分。这部分设计内容涵盖了一个完

此为试读，需要完整PDF请访问：www.ertongbook.com

整的软件工程过程：需求分析（需求建模）、概要设计（设计建模）、模块设计、模块实现、调试、发布。和一般软件工程过程不同的是上述每一步中都需要考虑对实时性的满足，因此更加复杂、也许需要专门通过一本书的篇幅对此进行探讨。我们只简单分析设计过程中实时多任务设计需要面临的关键问题：多任务划分、多任务分配、多任务调度。“关键”是因为它们是决定系统实时性的主要因素，具体设计时还存在其他一些问题，如任务间通信机制选择，中断服务程序设计等，都对系统实时性产生重大影响，但不属于本质的问题。对多任务划分、多任务分配和多任务调度的设计对应软件工程过程的概要设计和模块设计阶段。

多任务划分即如何将整个系统功能设计为不同的任务来实现，任务之间采取怎样的耦合关系，划分的粒度如何等。这里，“任务”也包括中断，因此也包括将什么功能放在中断中实现，什么功能放在常规的任务中处理。在根据数据流划分任务时，影响划分的要素包括数据流之间的并行和串行关系；根据控制流划分任务时，考虑的要素是控制的因果关系。

多任务划分影响着多任务分配和多任务调度。**多任务分配**决定任务放在哪个处理器上完成（存在多个处理器时），以及网络环境下任务如何分配。

多任务划分目的的实现需要**多**任务调度，**多任务调度**的设计目的是关键任务得到实时响应，同时整体上所有任务的设计内容都在允许的时间内完成。多任务调度的内容包括系统调度策略的选择，任务优先级的确定，以及任务间竞争和合作的设计。在划分多任务时，已经考虑了各任务所担任职责的轻重缓急，多任务调度时需要根据这种紧急程度分配合理的优先级；调度还必须使不同优先级的协作任务有效地同步。

多任务划分、多任务分配和多任务调度三者是有机的整体，而多任务划分则是其中决定性的部分。任何一个因素设计不合理都将影响整个系统的实时性。

1.1.2 微内核

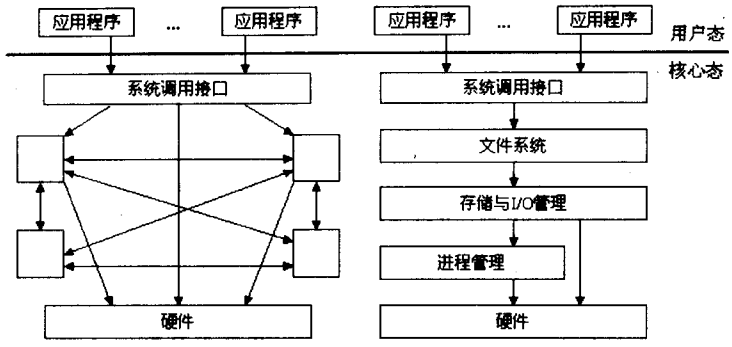
传统上，一个操作系统分为核心态和用户态。内核在核心态运行，为用户态的应用程序服务。内核是操作系统的灵魂和中心，决定了操作系统的效率和应用领域。在设计操作系统时，内核包含哪些功能以及内核功能采取何种组织结构，都是由设计者决定的。我们从内核功能和结构特点看，具有**整体式内核、层次式内核、微内核**三种不同形式。

整体式内核结构的操作系统实质上“无结构”。操作系统功能由一系列模块堆砌而成，任何模块之间可进行任意调用。整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。CP/M 和 MS-DOS 属于此类结构的操作系统。

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，VAX/VMS，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。两种操作系统的内核结构如图 1-1 所示。

不管整体式结构，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**操作系统，以至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。



（a）整体式结构　　（b）层次式结构

图 1-1　内核结构

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。

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程识别；
- 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；
- 适合分布式计算环境。内核为进程传递消息的方式天然适合 RPC 这一计算模式。

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 1-2 所示的 wind 微内核结构。

微内核的不同实现模式

从宏内核系统到微内核，操作系统结构发生了根本性的变化，导致的影响是多方面的。除了上面说明的微内核的优越性外，微内核的批评者也指出了微内核的缺陷：在微内核操作系统中，由于许多传统的操作系统功能改为用户任务实现，一般采取客户/服务器模型，即传统操作系统中许多系统功能作为微内核结构下的服务器任务，这样使任务间的数据交

换量更大，需要在内核与任务之间进行大量的数据复制，因此影响了系统性能。有研究人员指出著名的微内核操作系统 Mach 性能远不如宏内核的 BSD UNIX（当然这里与应用类型有关，例如用嵌入式微内核的系统实现一个文件服务器显然比 UNIX 效率低）。

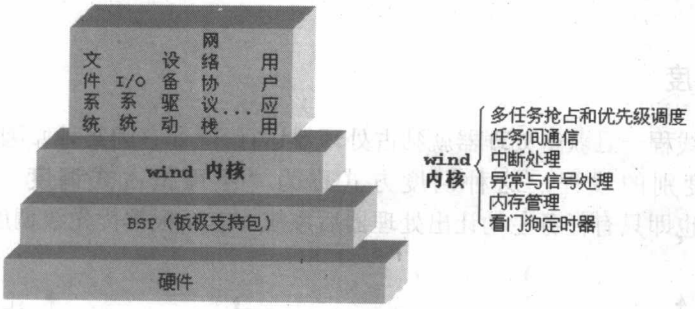


图 1-2 wind 微内核结构

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）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 微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。

系统调用时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个 trap 中断处理完成的。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。但是我们仍然沿用一般说法。

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。我们倾向于认为，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。有时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，可靠的系统由可靠的操作系统和可靠的应用系统组成）。对于从“单片机＋汇编语言”成长起来的开发人员，可能更喜欢单一实地址空间的系统。

1.1.3 任务调度

实时系统和分时系统的一个显著差异体现在调度策略上。实时系统调度关心的是对实

时事件的相应延迟，而传统的分时系统调度时要考虑的目标是多方面的：公平、效率、利用率、吞吐量等。因此实时系统通常采用优先级调度，即操作系统总是从就绪任务队列中选择最高优先级运行。优先级调度根据调度的时机又分为“不可抢占式调度”和“可抢占式调度”。

1．优先级调度

如果一个线程一旦获得处理器就独占处理器运行，除非它因某种原因决定放弃处理器，系统才会调度别的线程，这种调度方式称为“**不可抢占式调度**”（Non-Preemptive Scheduling），也即只有任务主动让出处理器后系统才重新根据优先级调度选择任务，如图1-3所示。

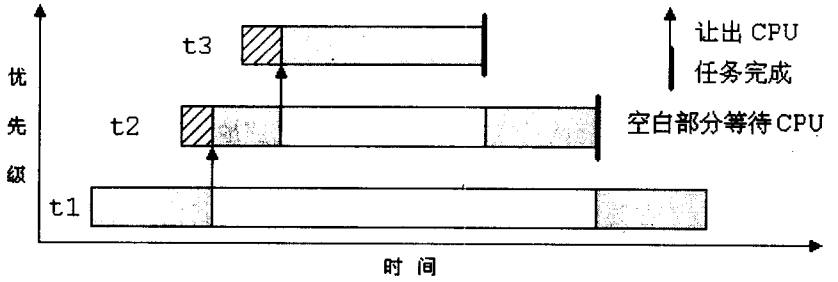


图 1-3　不可抢占优先级调度

在图1-3中，高优先级任务t2和t3就绪后（图中斜线阴影部分），并没有立即被调度，而是低优先级让出处理器时，才根据优先级高低选择任务运行。高优先级任务和低优先级任务之间是紧密合作的关系，低优先级任务必须设计为不使高优先级任务长时间等待。

当注重实时响应时，应该采用“**可抢占式调度**”（Preemptive Scheduling）。只要有更高优先级任务就绪，系统立即中断当前任务来调度高优先级任务，确保任意时刻最高优先级任务得到处理器，如图1-4所示。

在图1-4中，抢占式优先级调度使任务t2和t3就绪时总能立即得到处理器，其实时响应特性优于不可抢占调度。

对于区分核心态和用户态的系统，优先级调度还是微内核实现的基础。由于可抢占式内核中的核心态任务也可以随时让位给比其优先级高的任务，使得系统可以把一些实时设备的操作放在内核之外，通过“任务”完成，从而简化了内核设计。在传统的操作系统（如UNIX）中，必须将所有对时间要求较高的操作放在内核中实现，导致庞大的内核。

wind内核支持256级优先级：0~255。优先级0为最高优先级，优先级255为最低优先级。任务优先级在创建时确定，并允许程序运行中动态修改。但是对于内核而言，从就绪队列中选择一个任务调度时优先级是确定的，换言之，内核不会动态计算每个任务的优先级，因此这种调度策略属于**静态调度策略**；相对地，**动态调度策略**调度时需要根据某个目标（例如任务完成时间底线）动态确定任务优先级并调度。静态调度策略效率显著高于动态调度策略，并且足够满足应用需要，因而被普遍采用，包括POSIX 1003.1b标准对任

务调度的定义。

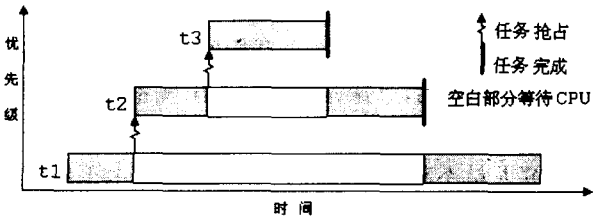


图 1-4 抢占式优先级调度

2. Round-Robin 调度

前面介绍的优先级调度存在这样的问题：如果没有被更高优先级任务抢占，或者因阻塞等原因让出处理器，任务将一直运行下去，在此情况下，同优先级任务将得不到运行。**Round-Robin 调度**基于这样的哲学：在更高优先级任务调度依然优先运行的前提下（这一点和前面一样），同优先级任务之间调度时追求一定意义上的公平。

Round-Robin 调度将任务运行划分为时间片，当任务运行一个时间片后，内核将其调出处理器并放在同优先级就绪任务队列尾部；调度时选择最高优先级就绪队列首部任务。Round-Robin 调度的效果是将每个任务运行一个时间片后“让出”处理器给下一个任务，如轮转一样，也称**轮转调度**。可见，Round-Robin 调度并没有改变“基于优先级”和“可抢占”这两个实时调度的特征。说在 VxWorks 中，Round-Robin 调度是基于优先级可抢占调度的一个“附加特征”，如图 1-5 所示。

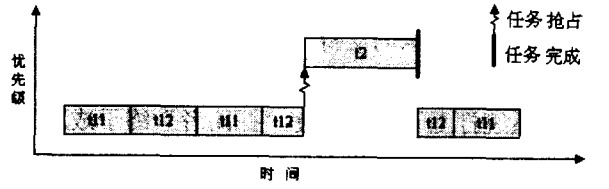


图 1-5 Round-Robin 调度

在 POSIX 1003.1b 中，基于优先级的可抢占调度定义为 **SCHED_FIFO**；Round-Robin 调度定义为 **SCHED_RR**。3.5 节“任务调度”部分深入介绍任务调度细节。

3. 优先级翻转问题

实时系统中，如果任务调度采用基于优先级的方式，**则传统的资源共享访问机制**在系统运行时很容易造成优先级翻转问题，即当一个高优先级任务访问共享资源时，该资源已被一低优先级任务占有，而这个低优先级任务在访问共享资源时可能又被其他一些低优先级的任务抢先，因此造成高优先级任务被许多较低优先级的任务阻塞，高优先级任务在低优先级任务之后运行，看起来像低优先级任务抢占了高优先级任务，即发生了**优先级翻转**（Priority Inversion）。

一个最为著名的优先级翻转问题的例子是“火星探路者”（Pathfinder），它采用了VxWorks实时内核设计。1997年7月4日，“探路者”弹出气囊在火星表明登陆，放出“漫游者”探测车，收集传回地球的大量数据。在开始的几天里，“探路者”被誉为完美无缺的作品，引起巨大轰动。但是几天后，“探路者”开始出现系统复位、数据丢失的现象。当时解释为“软件小故障”、“同时处理的事情太多”等。

为了找出故障，研究人员不断模拟“探路者”在火星上工作的条件和过程，终于在实验室中再现了系统复位的现象，并记录下来。根据分析，有如下两个任务需要互斥访问共享“信息总线”：（1）总线管理任务，具有最高优先级，运行频繁，进行总线数据I/O；（2）气象数据收集任务，优先级低，运行较少，收集数据并通过互斥信号量将数据发布到“总线”。如果数据收集任务持有信号量期间，总线管理任务就绪并且也申请获取信号量，则总线管理任务阻塞，直到收集任务释放信号量。看起来工作得很好，因为收集任务很快就会完成，高优先级的总线管理任务会很快得到运行。但是，另有一个需要较长时间运行的通信任务，其优先级比总线管理任务低，但是比数据收集任务高。在很少的情况下，如果通信被中断激活，并刚好在总线管理任务等待数据收集任务完成期间就绪，它将被系统调度，从而比它优先级低的数据收集任务得不到运行，并因此使最高优先级的总线管理任务无法运行。在经历较长时间后，看门狗观测到“总线”没有活动，将此解释为严重错误并使整个系统复位。

很显然需要防止优先级翻转以确保系统的实时响应。常见的解决办法之一是使用**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。

VxWorks实现了对优先级继承的支持，但是默认情况下该功能被关闭，也就是说有可能发生优先级翻转问题，也就导致了“探路者”中问题的发生。不过，研究人员及时查出了问题并给在火星上的系统打上了补丁，使“探路者”终于成功完成使命。美国航空航天局JPL（喷气推进实验室）决定第二代火星探测器项目仍然与风河系统进行合作，被引为VxWorks功能强大和可靠性的例证。

防止优先级翻转的另外一种协议是**优先级天花板**（Prioceiling），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看作的优先级，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。VxWorks对POSIX信号量的支持实现了该协议。

我们后面将对优先级翻转问题做更具体的描述。

1.2 任务属性

VxWorks任务具有两个显著不同于主机操作系统的特点：（1）VxWorks任务和内核具

此为试读，需要完整PDF请访问：www.ertongbook.com