VxWorks操作系统指南

目　录

关键词：实时操作系统　　任务　消息　VxWorks　　Tornado

摘　　要：本文档对实时操作系统作了简要介绍，并针对VxWorks系统的特点进行了具体的说明和分析，从VxWorks系统的任务管理、通信机制、系统配置、系统接口几个方面展开。

1. VxWorks操作系统概述

1.1.VxWorks 操作系统简介

实时多任务操作系统是能在确定的时间内执行其功能，并对外部的异步事件作出响应的计算机系统。多任务环境允许一个实时应用作为一系列独立任务来运行，各任务有各自的线程和系统资源。VxWorks系统提供多处理器间和任务间高效的信号灯、消息队列、管道、网络透明的套接字。 实时系统的另一关键特性是硬件中断处理。为了获得最快速可靠的中断响应，VxWorks系统的中断服务程序ISR有自己的上下文。

VxWorks实时操作系统由400多个相对独立的、短小精炼的目标模块组成，用户可根据需要选择适当模块来裁剪和配置系统，这有效地保证了系统的安全性和可靠性。系统的链接器可按应用的需要自动链接一些目标模块。这样，通过目标模块之间的按需组合，可得到许多满足功能需求的应用。

VxWorks操作系统的基本构成模块包括以下部分：

- 高效的实时内核Wind

VxWOrks实时内核（Wind）主要包括基于优先级的任务调度、任务同步和通信、中断处理、定时器和内存管理。

- 兼容实时系统标准POSIX

VxWOrks提供接口来支持实时系统标准P.1003.1b.

- I/O系统

VxWOrks提供快速灵活的与ANSI-C相兼容的I/O系统，包括UNIX的缓冲I/O和实时系统标准POSIX的异步I/O。VxWOrks包括以下驱动：

网络　　---　网络设备（以太网、共享内存）

管道　　---　任务间通信

RAM　　---　驻留内存文件

SCSI　　---　SCSI硬盘，磁碟，磁带

键盘　　---　PC x86键盘（BSP仅支持x86）

显示器 --- PC x86显示器（BSP仅支持x86）

磁碟 --- IDE和软盘（BSP仅支持x86）

并口 --- PC格式的目标硬件

- 本机文件系统
- I/O系统

VxWorks的文件系统与MS-DOS、RT-11、RAM、SCSI等相兼容。

- 网络特性

VxWorks网络能与许多运行其它协议的网络进行通信，如TCP/IP、4.3BSD、NFS、UDP、SNMP、FTP等。VxWorks可通过网络允许任务存取文件到其它系统中，并对任务进行远程调用。

- 虚拟内存（可选单元VxVMI)

VxVMI主要用于对指定内存区的保护，如内存块只读等，加强了系统的健壮性。

- 共享内存（可选单元VxMP)

VxMP主要用于多处理器上运行的任务之间的共享信号量、消息队列、内存块的管理。

- 驻留目标工具

Tornado集成环境中，开发工具工作于主机侧。驻留目标外壳、模块加载和卸载、符号表都可进行配置。

- Wind基类

VxWorks系统提供对C++的支持，并构造了系统基类函数。

- 工具库

VxWorks系统向用户提供丰富的系统调用，包括中断处理、定时器、消息注册、内存分配、字符串转换、线性和环形缓冲区管理，以及标准ANSI-C程序库。

- 性能优化

VxWorks系统通过运行定时器来记录任务对CPU的利用率，从而进行有效地调整，合理安排任务的运行，给定适宜的任务属性。

- 目标代理

目标代理可使用户远程调试应用程序。

- 板级支持包

板级支持包提供硬件的初始化、中断建立、定时器、内存映象等。

- VxWorks仿真器（VxSim)

可选产品VxWorks仿真器，能模拟VxWorks目标机的运行，用于应用系统的分析。

1.2.VxWorks操作系统内核

VxWorks内核（wind）的基本功能可以分为如下几大类：

- 任务管理；
- 事件和异步信号服务；
- 信号量服务；
- 消息队列服务；
- 内存管理；
- 中断服务程序；
- 时钟管理和定时器服务；
- 出错处理。

在以下各节中将对VxWorks内核的各类功能分别进行描述。

1.1.任务管理

任务是代码运行的一个映象，从系统的角度看，任务是竞争系统资源的最小运行单元。任务可以使用或等待CPU、I/O设备及内存空间等系统资源，并独立于其它任务，与它们一起并发运行（宏观上如此）。VxWorks内核使任务能快速共享系统的绝大部分资源，同时有独立的上下文来控制个别线程的执行。

1.1.1.任务结构

多任务设计能随时打断正在执行着的任务，对内部和外部发生的事件在确定的时间里作出响应。VxWorks实时内核Wind提供了基本的多任务环境。从表面上来看，多个任务正在同时执行，实际上，系统内核根据某一调度策略让它们交替运行。系统调度器使用任务控制块的数据结构（简记为TCB)来管理任务调度功能。任务控制块用来描述一个任务，每一任务都与一个TCB关联。TCB包括了任务的当前状态、优先级、要等待的事件或资源、任务程序码的起始地址、初始堆栈指针等信息。调度器在任务最初被激活时以及从休眠态重新被激活时，要用到这些信息。

此外，TCB还被用来存放任务的"上下文"（context)。任务的上下文就是当一个执行中的任务被停止时，所要保存的所有信息。在任务被重新执行时，必须要恢复上下文。通常，上下文就是计算机当前的状态，也即各个寄存器的内容。如同在发生中断所要保存的内容一样。当发生任务切换时，当前运行的任务的上下文被存入TCB，将要被执行的任

务的上下文从它的TCB中取出，放入各个寄存器中。于是转而执行这个任务，执行的起点是前次它在运行时被中止的位置。

VxWorks中，内存地址空间不是任务上下文的一部分。所有的代码运行在同一地址空间。如每一任务需各自的内存空间，需可选产品VxVMI的支持。

1.1.2.任务状态和状态迁移

实时系统的一个任务可有多种状态，其中最基本的状态有四种：

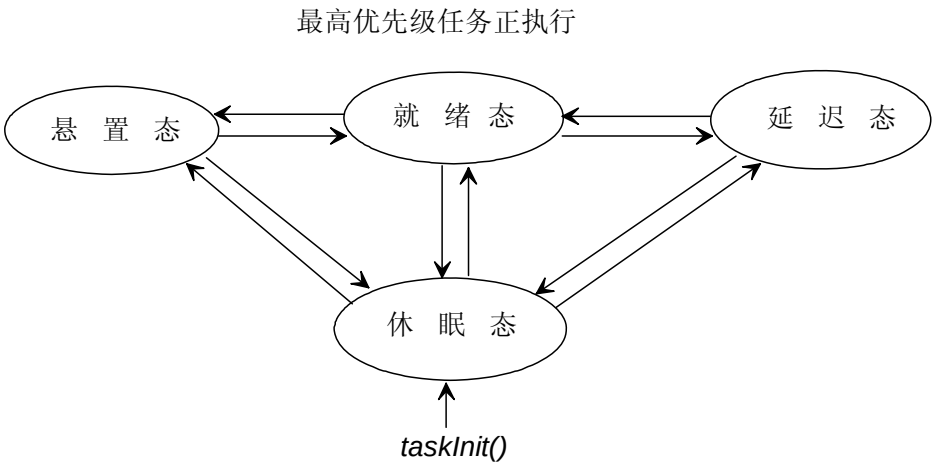
就绪态：任务只等待系统分配CPU资源；

悬置态：任务需等待某些不可利用的资源而被阻塞；

休眠态：如果系统不需要某一个任务工作，则这个任务处于休眠状态；

延迟态：任务被延迟时所处状态；

当系统函数对某一任务进行操作时，任务从一种状态迁移到另一状态。处于任一状态的任务都可被删除。



状态迁移	调用
就绪态 ----> 悬置态	
semTake()/msgQReceive()	
就绪态 ----> 延迟态	*taskDelay()*
就绪态 ----> 休眠态	*taskSuspend()*
悬置态 ----> 就绪态	*semGive()/msgQSend()*
悬置态 ----> 休眠态	*taskSuspend()*
延迟态 ----> 就绪态	expired delay
延迟态 ----> 休眠态	*taskSuspend()*

休眠态 ----> 就绪态

taskResume()/taskActivate()

休眠态 ----> 悬置态 *taskResume()*

休眠态 ----> 延迟态 *taskResume()*

1.1.3.任务调度策略

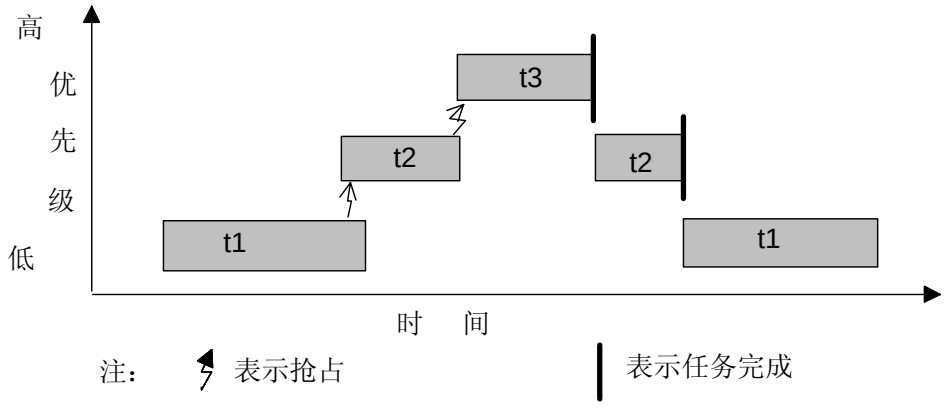
多任务调度须采用一种调度算法来分配CPU给就绪态任务。Wind内核采用基于优先级的抢占式调度法作为它的缺省策略，同时它也提供了轮转调度法。

基于优先级的抢占式调度，它具有很多优点。这种调度方法为每个任务指定不同的优先级。没有处于悬置或休眠态的最高优先级任务将一直运行下去。当更高优先级的任务由就绪态进入运行时，系统内核立即保存当前任务的上下文，切换到更高优先级的任务。

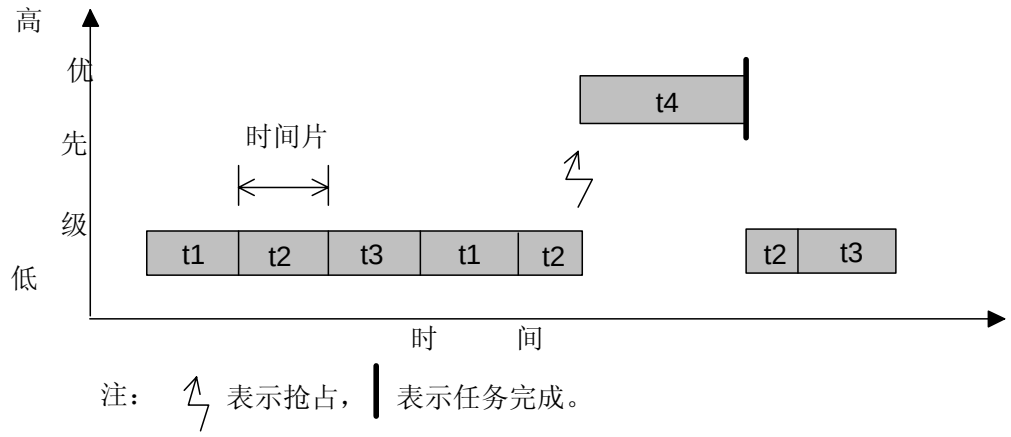
多任务调度须采用一种调度算法来分配CPU给就绪态任务。Wind内核采用基于优先级的抢占式调度法作为它的缺省策略，同时它也提供了轮转调度法。

基于优先级的抢占式调度，它具有很多优点。这种调度方法为每个任务指定不同的优先级。没有处于悬置或休眠态的最高优先级任务将一直运行下去。当更高优先级的任务由就绪态进入运行时，系统内核立即保存当前任务的上下文，切换到更高优先级的任务。

Wind内核划分优先级为256 级（0~255）。优先级0为最高优先级，优先级255为最低。当任务被创建时，系统根据给定值分配任务优先级。然而，优先级也可以是动态的，它们能在系统运行时被用户使用系统调用taskPrioritySet()来加以改变，但不能在运行时被操作系统所改变。



轮转调度法分配给处于就绪态的每个同优先级的任务一个相同的执行时间片。时间片的长度可由系统调用KernelTimeSlice()通过输入参数值来指定。很明显，每个任务都有一运行时间计数器，任务运行时每一时间滴答加1。一个任务用完时间片之后，就进行任务切换，停止执行当前运行的任务，将它放入队列尾部，对运行时间计数器置零，并开始执行就绪队列中的下一个任务。当运行任务被更高优先级的任务抢占时，此任务的运行时间计数器被保存，直到该任务下次运行时。



1.1.4.抢占禁止

Wind内核可通过调用*taskLock()*和*taskUnlock*（）来使调度器起作用和失效。当一个任务调用*taskLock()*使调度器失效，任务运行时没有基于优先级的抢占发生。然而，如果任务被阻塞或是悬置时，调度器从就绪队列中取出最高优先级的任务运行。当设置抢占禁止的任务解除阻塞，再次开始运行时，抢占又被禁止。这种抢占禁止防止任务的切换，但对中断处理不起作用。

1.1.5.异常处理

程序代码和数据的出错，如非法命令、总线或地址错误、被零除等。VxWorks异常处理包，一般是将引起异常的任务休眠，保存任务在异常出错处的状态值。内核和其它任务继续执行。用户可借助Tornado开发工具，查看当前任务状态，从而确定被休眠的任务。

1.1.6.任务管理

VxWorks内核的任务管理提供了动态创建、删除和控制任务的功能，具体实现通过如下一些系统调用：

taskSpawn()	创建（产生并激活）新任务
taskInit()	初始化一个新任务
taskActivate()	激活一个已初始化的任务
taskName()	由任务ID号得到任务名
taskNameToId（）	由任务名得到任务ID号
taskPriorityGet()	获得任务的优先级
taskIsSuspended()	检查任务是否被悬置
taskIsReady()	检查任务是否准备运行
taskTcb()	得到一个任务控制块的指针
taskDelete()	中止指定任务并自由内存（仅任务堆栈和控制块）
taskSafe()	保护被调用任务
taskSuspend()	悬置一个任务
taskResume()	恢复一个任务
taskRestart()	重启一个任务
taskDelay()	延迟一个任务

1.2.通信、同步和互斥机制

VxWorks支持各种任务间通信机制，提供了多样的任务间通信方式，主要有如下几种：

- 共享内存，主要是数据的共享；
- 信号量，用于基本的互斥和任务同步；
- 消息队列和管道，单CPU的消息传送；
- Socket和远程过程调用，用于网络间任务消息传送；
- 二进制信号，用于异常处理。

在多处理器之间的任务也可采用共享内存对象来实现任务间通信,只是在系统配置上有所不同.

1.1.1.共享存储区

任务间通信的最简单的方法是采用共享存储区，也即相关的各个任务分享属于它们的地址空间的同一内存区域。因为所有任务都存在于单一的线性地址空间，任务间共享数据。全局变量、线性队列、环形队列、链表、指针都可被运行在不同上下文的代码所指向。

1.1.2.互斥

当某一地址空间用于数据交换时，为了避免冲突，对于内存的锁定是非常重要的。两个或多个任务读写某些共享数据时，最后的结果取决于任务运行的精确时序，有可能得到错误值，这样必须以某种手段确保当一个任务在使用一个共享变量或文件时，其他任务不能做

同样的操作。主要有关中断、抢占禁止和用信号量锁定资源等方法。一般来说，关中断是最有效的解决互斥的方法。但这对于实时应用来说，它阻止系统对外部事件的响应，无法满足实时性的要求。同样，中断延迟也是不能接受。

1.1.3.信号量

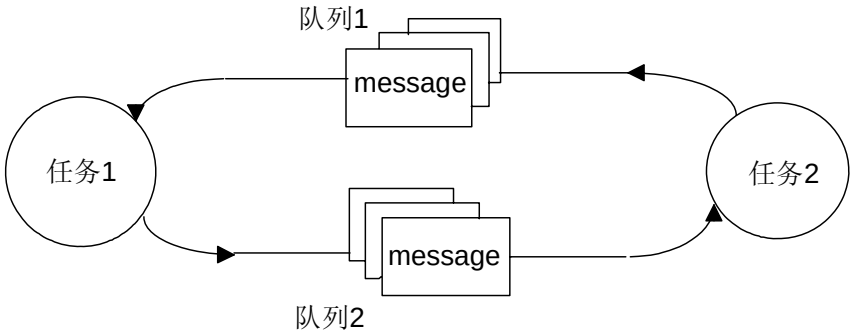
VxWorks信号量提供最快速的任务间通信机制，它主要用于解决任务间的互斥和同步。针对不同类型的问题，有以下三种信号量：

- 二进制信号量　　使用最快捷、最广泛，主要用于同步或互斥；
- 互斥信号量　　主要用于优先级继承、安全删除和回溯；
- 计数器

VxWorks还提供POSIX信号量和多处理器上信号量的应用。

1.1.1.消息队列

现实的实时应用由一系列互相独立又协同工作的任务组成。信号量为任务间同步和联锁提供了高效方法。单处理器中任务间消息的传送采用消息消息队列。消息机制使用一个被各有关进程共享的消息队列，任务之间经由这个消息队列发送和接收消息。



任务间全双工信息传送

1.1.2.管道

管道用VxWorks的I/O系统提供一种灵活的消息传送机制，它是受驱动器pipeDrv（VxWorks所提供）管理的虚拟I/O设备。任务能调用标准的I/O函数打开、读出、写入管道。当任务试图从一个空的管道中读取数据，或向一个满的管道中写入数据时，任务被阻塞。和消息队列类似，ISR能向管道中写入信息，但不能从中读取。象I/O设备一样，

管道有一个消息队列所没有的优势----调用select()，任务等待一系列I/O设备上的数据。

1.1.3.系统实现

Wind信号量对于各种类型的信号量的控制提供了同一规范化的接口，仅创建函数要特别指明信号量类型。

semBCreate()	创建（产生并激活）一个二进制信号量
semMCreate()	创建（产生并激活）一个互斥信号量
semCCreate()	创建（产生并激活）制一个计数信号量
semDelete()	中止并自由信号量
semTake（）	获得信号量
semGive()	给出信号量
semFlush()	解锁所有正等待某一信号量的任务

Wind 消息队列管理：

msgQCreate()	创建（产生并激活）消息队列
msgQDelete()	中止并自由信号量
msgQSend()	向消息队列发送消息
msgQReceive()	从消息队列接收消息

1.2.网络通信

VxWorks提供了强大的网络功能，能与其它许多主机系统进行通信。网络完全兼容4.3BSD，也兼容SUN公司的NFS。这种广泛的协议支持在主机和VxWorksh目标机之间提供了无缝的工作环境,任务可通过网络向其它系统的主机存取文件，即远程文件存取，也支持远程过程调用。通过以太网，采用TCP/IP和UDP/IP协议在不同主机之间传送数据。

VxWorks提供了如下一些网络工具完成信息传送：

- Sockets

完成运行在VxWorks系统或其它系统之间任务的消息传送；

- 远程过程调用（RPC）

允许任务调用另一主机（运行的系统为VxWorks或是其它）上的过程。

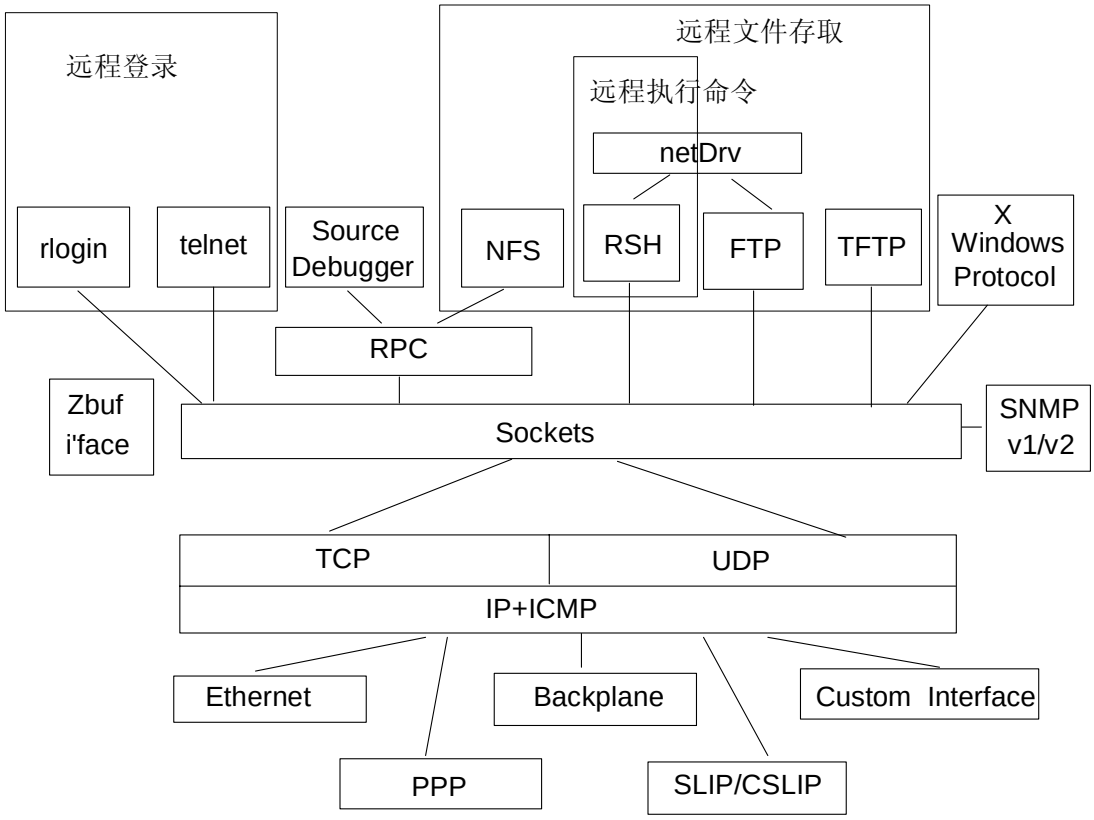
- 远程文件存取

VxWorks任务可采用NFS、RSH、FTP、TFTP等方式远程存取主机文件。

- 文件输出
- 远程执行命令

VxWorks任务可通过网络激活主机系统中的命令。

VxWorks网络组件结构如下：

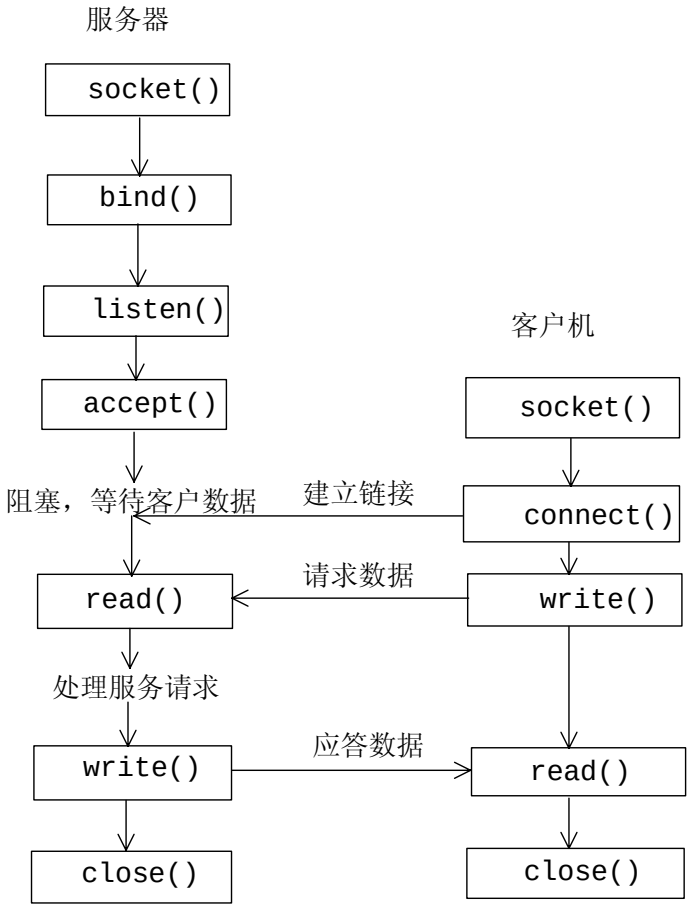


1.2.1.套接口（Sockets）

Vxworks系统和网络协议的接口是靠套接字（sockets）来实现的。Sockets规范是得到广泛应用的、开放的、支持多种协议的网络编程接口。通讯的基石是套接口，一个通讯口是套接的一端，在这一端上你可以找到其对应的一个名字。一个正在被使用的套接口都有它的类型和与其相关的任务。套接口存在于通讯域中。通讯域是为了处理一般的线程通过套接口通讯而引进的一种抽象概念。套接口通常和同一个域中的套接口交换数据（数据交换也可能穿越域的界限，但这时一定要执行某种解释程序）。各个任务使用这个域互相之间用Internet协议来进行通讯。

套接口可以根据通讯性质分类。应用程序一般仅在同一类的套接口间通讯。不过只要底层的通信协议允许，不同类型的套接口间也照样可以通信。用户目前使用两种套接口，即流套接口（采用TCP协议）和数据报套接口（采用UDP协议）。流套接口提供了双向的、有序的、无重复并且无记录边界的数据流服务。数据报套接口支持双向的数据流，但并不保证是可靠、有序、无重复的。也就是说，一个从数据报套接口接收信息的任务有可能发现信息重复了，或者和发出时的顺序不同。数据报套接口的一个重要特点是它保留了记录边界。对于这一特点，数据报套接口采用了与现在许多包交换网络（例如以太网）非常类似的模型。

套接口（socket)通信的最大优点是：过程间的通信是完全对等的，不管网络中过程的定位或主机所运行的操作系统。一般来说，流套套接口提供了可靠的面向连接的服务，应用较广泛。其应用程序时序图如下：



Socket函数：

socket()	创建一个套接口
bind()	给套接口分配名称
listen()	打开TCP套接口连接
accept()	完成套接口间连接
connect()	请求连接套接口
shutdown()	关闭套接口间连接
send()	向TCP套接口发送数据
recv()	从TCP套接口接收数据
select()	完成同步I/O传输
read()	从套接口读取信息
write()	向套接口写入信息
ioctl()	完成对套接口的控制
close()	关闭套接口

1.1.中断服务程序

实时系统中硬件中断处理是至关重要的，因为它是以中断方式通知系统外部事件的发生。为了快速响应中断，中断服务程序ISR运行在特定的空间，不同于其它任何任务，因此中断处理没有任务的上下文切换。

中断函数：

intConnect()	将C函数和中断向量联结
intCount()	得到当前中断套叠深度
intLevelSet()	设置程序中断级别
intLock()	使中断禁止
intUnlock()	开中断
intVecSet()	设置异常向量
intVecGet()	得到异常向量

所有的中断服务程序使用同一中断堆栈，它在系统启动时就已根据具体的配置参数进行了分配和初始化，必须保证它的大小，以使它能满足最坏的多中断情况。中断也有缺陷：ISR不运行在常规的任务上下文，它没有任务控制块。对于ISR的基本约束就是它们不能激活那些可能使调用程序阻塞

的函数，例如，它不能获取信号量，因如果该信号量不可利用，内核会试图让调用者切换到悬置态。然而，ISR能给出信号量。

一个ISR通常与一个或多个任务进行通信，有直接的也有间接的作为输入输出事务的一部分。这种通信的本质是驱动任务执行,从而处理中断和各种情况。这与任务到任务的通信和同步基本相同，但是有两点不同：

- 一个ISR通常作为通信或同步的发起者，它通常返回一个信号量、向队列发送一个信息包或事件给一个任务。ISR很少作为信息的接收者，它不可以等待接收信息包或事件。
- ISR内的系统调用总是立即返回ISR本身。例如，即使ISR通过发送信息包唤醒了一个很高优先级的任务，它也首先必须返回ISR。这是因为ISR必须先完成。

1.1.时钟管理

时钟管理提供以下功能：

- 维护系统日历时钟；
- 在任务等待消息包、信号量、事件或内存段时的超时处理；
- 以一定的时间间隔或在特定的时间唤醒或发送告警到一个任务。
- 处理任务调度中的时间片轮循。

这些功能都依赖于周期性的定时中断，离开实时时钟或定时器硬件就无法工作。

时钟管理的系统调用有：

tickAnnounce()	通知系统内核时钟“滴答”
tickSet()	设定内核时钟计数器值
tickGet()	得到内核时钟计数器值
timer_create()	创建时钟
timer_gettime()	获得时钟器给定值的当前剩余值
timer_settime()	设定时钟值
timer_connect()	联系用户函数和时钟信号
timer_cancel()	取消一个时钟
sysClkRateSet()	系统时钟速率设置

VxWorks看门狗定时器作为系统时钟中断服务程序的一部分，允许C语言函数指明某一时间延迟。一般来说，被看门狗定时器激活的函数运行在系统时钟中断级。然而，如果内核不能立即运行该函数，函数被放入tExcTask工作队列中。在tExcTask工作队列中的任务运行在最高优先级0。

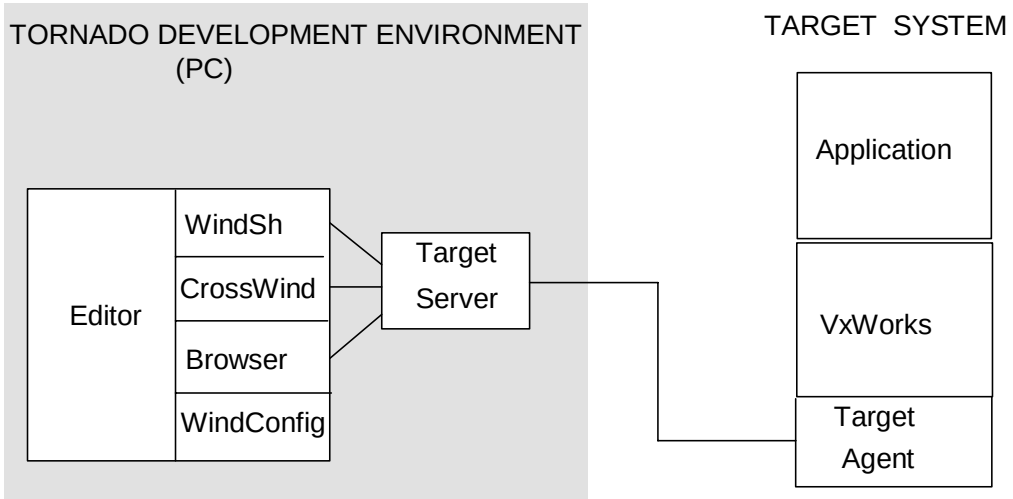
看门狗定时器调用函数：

wdCreate()	分配并初始化看门狗定时器
wdDelete()	中止并解除看门狗定时器
wdStart()	启动看门狗定时器
wdCancel()	取消当前正在计数的看门狗定时器

1. VxWorks应用指导

Tornado集成环境提供了高效明晰的图形化的实时应用开发平台，它包括一套完整的面向嵌入式系统的开发和调测工具。Tornado环境采用主机--目标机交叉开发模型，应用程序在主机的Windows环境下编译链接生成可执行文件，下载到目标机，通过主机上的目标服务器与目标机上的目标代理程序的通信完成对应用程序的调测、分析。它主要由以下几部分组成：

- VxWorks,高性能的实时操作系统；
- 应用编译工具；
- 交互开发工具；



Tornado：交互主机工具

下面对Tornado集成环境的各组件功能分别介绍：

- Tornado开发环境

Tornado是集成了编辑器、编译器、调试器于一体的高度集成的窗口环境，同样也可以从Shell窗口下发命令和浏览。

- WindConfig：Tornado系统配置

通过WindConfig可选择需要的组件组成VxWorks实时环境，并生成板级支持包BSP的配置。

- WindSh:Tornado外壳

WindSh是一个驻留在主机内的C语言解释器，通过它可运行下载到目标机上的所有函数，包括VxWorks和应用函数。Tornado外壳还能解释常规的工具命令语言TCL。

- 浏览器

Tornado浏览器可查看内存分配情况、系统目标（如任务、消息队列、信号量等）。这些信息可周期性地进行更新。

- CrossWind:源码级调试器

源码级调试器CrossWind提供了图形和命令行方式来调试，可进行指定任务或系统级断点设置、单步执行、异常处理。

- 驻留主机的目标服务器

目标服务器管理主机与目标机的通信，所有与目标机的交互工具都通过目标服务器，它也管理主机上的目标机符号表，提供目标模块的加载和卸载。

- Tornado注册器

所有目标服务器注册其提供的服务在注册器中。注册器映射用户定义的目标名到目标服务器网络地址。

- VxWOrks

Tornado包含了VxWorks操作系统。

- 目标代理程序

目标代理程序是一个驻留在目标机中的联系Tornado工具和目标机系统的组件。一般来说，目标代理程序往往是不可见的。

1.1.系统启动

1.1.1.启动盘的制作

在实时应用系统的开发调测阶段，往往采用以PC机作为目标机来调测程序。主机PC和目标机PC之间可采取串口或是网口进行联结。由于大多数目标已配有网卡，网络联结成为最简单快速的连接方式。串口联结虽通信速率不高，也有它自己的优点，系统级任务调试（如中断服务程序ISR）需使通信方式工作在Polled 模式，网口联结就不支持，因此可以裁剪掉系统中网络部分，以使VxWorks系统更小，满足目标板的内存约束。下面分别对这两种通信方式下目标机VxWorks系统启动盘的制作作一简要介绍。

串口通信时目标机VxWorks系统启动盘的制作步骤：

1.修改通用配置文件\\Tornado\target\config\pc486\config.h.

在config.h文件中加入以下宏定义：

```
#undef      WDB_COMM_TYPE
#define     WDB_COMM_TYPE     WDB_COMM_SERIAL   /*定义通信方式为串口联结*/
#define     WDB_TTY_CHANNEL   1                 /*通道号*/
#define     WDB_TTY_BAUD      9600              /*串口速率，可设置至38400*/
```

并且修改#define DEFAULT_BOOT_LINE中vxWorks为vxWorks.st。

2.在Tornado集成环境中点取Project菜单，选取Make PC486，选择Common Target,先进行clean操作;再选择Boot Rom Target,进行bootrom_uncmp操作；再选择VxWorks Target,进行vxworks.st操作。

3.拷贝\\Tornado\target\config\pc486\bootrom_uncmp至\\Tornado\host\bin下;

4.重命名文件bootrom_uncmp为bootrom;

5.准备一张已格式化的空盘插入软驱；

6.在目录\\Tornado\host\bin下执行命令 mkboot a: bootrom ；

7.拷贝\\Tornado\target\config\pc486\VxWorks.*至软盘;

8.将系统制作盘插入目标机软驱，加电启动目标机即载入VxWorkst系统。

网口通信时目标机VxWorks系统启动盘的制作步骤：

1.配置目标机网卡，设置其中断号和输入输出范围（I/O地址）；

2.修改通用配置文件\\Tornado\target\config\pc486\config.h.

针对不同的网卡，其名称不同，如NE2000及其兼容网卡为ENE，3COM以太网卡为ELT，Intel网卡为EEX。

在config.h文件中修改相应网卡类型（如网卡为3COM网卡）的定义部分：

```
#define     IO_ADRS_ELT       网卡I/O地址
```

#define INT_LVL_ELT 网卡中断号

并且修改#define DEFAULT_BOOT_LINE的定义：

#define DEFAULT_BOOT_LINE\

"elt(0,0)主机标识名:C:\\tornado\\target\\config\\pc486\\vxWorks h=主机IP e=目标机IP u=登录用户名 pw=口令 tn=目标机名"

3.主机信息的确定

主机操作系统Win95安装目录下有一文件hosts.sam，向其中加入：

主机IP 主机名

目标机IP 目标机名

4.在Tornado集成环境中点取Project菜单，选取Make PC486，选择Common Target,先进行clean操作;再选择Boot Rom Target,进行bootrom_uncmp操作；再选择VxWorks Target,进行vxworks操作。

5.拷贝\\Tornado\target\config\pc486\bootrom_uncmp至\\Tornado\host\bin下;

6.重命名文件bootrom_uncmp为bootrom;

7.准备一张已格式化的空盘插入软驱;

8.在目录\\Tornado\host\bin下执行命令 mkboot a: bootrom ；

9.启动Tornado组件FTP Server,在WFTPD窗口中选择菜单Security中的User/right...,在其弹出窗口中选择New User...,根据提示信息输入登录用户名和口令，并且要指定下载文件vxWorks所在根目录;还必选取主菜单Logging中Log options,使Enable Logging、Gets 、Logins 、Commands 、Warnings能。

10.将系统制作盘插入目标机软驱，加电启动目标机即通过FTP方式从主机下载VxWorkst系统。

1.1.1.主机Tornado环境配置

串口联结时主机Tornado开发环境的目标服务器配置操作如下：

1.在Tornado集成环境中点取Tools菜单，选取Target Server，选择config...;

2.在Configure Target Servers窗口中先给目标服务器命名;

3.在配置目标服务器窗口中的"Change Property"窗口中选择Back End,在"Available Back"窗口中选择wdbserial,再在"Serial Port"窗口中选择主机与目标机连接所占用的串口号（COM1,COM2),再在"Speed(bps)"窗口中选择主机与目标机间串口速率。

4. 在配置目标服务器窗口中的"Change Property"窗口中选择Core File and Symbols,选择File为BSP目标文件所在目录（本例为PC486目录）的VxWorks.st,并选取为All Symbols.

5.在配置目标服务器窗口中的"Change Property"窗口中的其它各项可根据需要选择。

网口联结时主机Tornado开发环境的目标服务器配置操作如下：

1.在Tornado集成环境中点取Tools菜单，选取Target Server，选择config...;

2.在Configure Target Servers窗口中先给目标服务器命名；

3.在配置目标服务器窗口中的"Change Property"窗口中选择Back End,在"Available Back" 窗口中选择wdbrpc,在"Target IP/Address"窗口中输入目标机IP。

4. 在配置目标服务器窗口中的"Change Property"窗口中选择Core File and Symbols, 选择File为BSP目标文件所在目录（本例为PC486目录）的VxWorks,并选取为All Symbols.

5.在配置目标服务器窗口中的"Change Property"窗口中的其它各项可根据需要选择。

1.1.应用系统配置

运行在目标板上的系统映象是个二进制模块。大多数情况下，用户会发现系统映象占用空间较大。然而，用户可根据需要裁剪系统配置，降低系统占用资源。

下面针对配置系统映象从以下几方面进行说明：

- VxWorks板级支持包（BSP）.
- VxWorks配置文件、可选项、参数.
- VxWorks某些可选择配置.

1.1.1.板级支持包BSP

Tornado目录下config/bspname包含板级支持包BSP，它由运行VxWorks的某些硬件驱动文件组成，如有串行线的VME板、时钟和其它设备。文件包括：**Makefile, sysLib.c,sysSerial.c,sysALib.s,romInit.s**,*bspname*.**h**,and **config.h**.

文件**sysLib.c**以硬件独立方式提供VxWorks和应用程序间的板级联系，包括：

- 初始化函数
  - 初始化硬件到一已知状态
  - 标识系统
  - 初始化设备，如SCSI或常规设备
- 内存/地址空间函数
  - 得到板上内存大小
  - 总线地址空间
  - 设定/获得非易失性RAM
  - 定义板的内存位图（可选）

- 为有MMU的处理器定义虚拟内存到物理内存的映射

• 总线中断函数

- 打开/关闭总线中断
- 产生总线中断

• 时钟/定时器函数

- 使能/不能定时中断
- 设置定时器的周期性

• 邮箱/位置监视函数（可选）

- 使邮箱/位置监视中断能

在目录config/all 的配置文件usrConfig.c和bootConfig.c负责启动库函数。设备驱动可调用内存和总线管理函数。

1.1.1.1.虚拟内存

对于支持MMU的单板，数据结构sysPhysMemDesc用来定义虚拟内存到物理内存的映射。该数据一般定义在sysLib.c中，也有的在一单独的文件memDesc.c中。它以数据结构PHYS_MEM_DESC的数组存在。sysPhysMemDesc数组记录用户的系统配置。

1.1.1.2.串行设备

文件sysSerial.c提供对目标板串口的初始化。实际的串口I/O设备在目录src/drv/sio下。ttyDrv库使用串口I/O设备提供VxWorks的终端操作。

1.1.1.3.初始化模块

romInit.s包括汇编级初始化程序，sysALib.s包含初始部分和具体系统的汇编级程序。

1.1.2.4配置VxWorks

VxWorks 的配置头文件为 config/all/configAll.h 和 config/bspname/config.h.当运行配置VxWorks的初始化时，这些文件被程序usrConfig.c\bootConfig.c\bootInit.c调用。在开发环境中，用户可能要测试几种不同的配置，或者用户想在不同情况下指明不同的目标代码。为了编译VxWorks满足不同情况，用户必须调整使用环境。

用户Tornado环境包括三部分：主机代码、目标代码和配置文件。缺省配置文件为：

Host code $WIND_BASE/host/hosttype/bin

Target code TGT_DIR=$WIND_BASE/target
Configuration code
CONFIG_ALL = TGT_DIR/config/all

用户可修改通用配置文件configAll.h和具体的目标板配置文件config.h。许多可选特性和设备驱动用户在文件config/all/usrConfig.c模块中可按需调整。

宏	选择
INCLUDE_ADA	Ada支持
INCLUDE_ANSI_XXX	各种ANSI C函数库选择
INCLUDE_BOOTP	BOOTP支持
INCLUDE_CACHE_SUPPORT	缓冲支持
INCLUDE_CPLUS	C++支持
INCLUDE_CPLUS_XXX	各种C++支持
INCLUDE_DEMO	使用简单的demo程序
INCLUDE_FTP_SERVER	FTP服务器支持
INCLUDE_HW_FP	硬件浮点支持
INLCUDE_LOADER	驻留目标机目标模块加载包
INCLUDE_LOGGING	注册工具
INCLUDE_MMU_BASIC	MMU支持
INCLUDE_MSG_Q	消息队列支持
INCLUDE_NETWORK	网络支持
INCLUDE_POSIX_XXX	各种POSIX选择
INCLUDE_RLOGIN	用rlogin远端注册
INCLUDE_RPC	远程过程调用
INCLUDE_SEM_BINARY	二进制信号量
INCLUDE_SEM_COUNTING	计数信号量
INCLUDE_SEM_MUTEX	互斥信号量
INCLUDE_SHELL	C语言解释器
INCLUDE_SPY	任务活动监视器
INCLUDE_WATCHDOGS	看门狗
INCLUDE_WDB	目标机代理

1.2.板级支持包BSP

板级支持包BSP负责目标板硬件的初始化，实时内核的载入等。

对于硬件初始化的顺序，大致可按下表中形式进行：

函　数	函　数　功　能	所 在 文 件
sysInit()	(a)锁住中断；(b)禁用缓冲； (c)用缺省值初始化系统中断表（仅i960); (d)用缺省值初始化系统错误表（仅i960); (e)初始化处理器寄存器到一缺省值; (f)使回溯失效；(g)清除所有悬置中断； (h)激活usrInit()，指明启动类型。	sysALib.s
UsrInit()	(a)对bss赋零； (b)保存bootType于sysStartType; (c)调用excVecInit(),初始化所有系统和缺省中断向量； (d)依次调用sysHwInit(), usrKernelInit(),kernelInit().	usrConfig.c
usrKernelInit()	依次调用 classLibInit(),taskLibInit(),taskHookInit(),semBLibInit(),semMLibInit(),semCLibInit(),semOLibInit(),wdLibInit(),msgQLibInit(),qInit(),workQInit()	usrKernel.c
kernelInit()	初始化并启动内核。 (a)激活intLockLevelSet(); (b)从内存池顶部创建根堆栈和TCB； (c)调用taskInit(),taskActivate(),用于usrRoot(); (d)调用usrRoot().	kernelLib.c
UsrRoot()	初始化I/O系统，驱动器，设备（在configAll.h和config.h中指定） (a)调用 sysClkConnect(),sysClkRateSet(), iosInit(),[ttyDrv()]; (b)初始化 excInit(),logInit(),sigInit(). (c)初始化管道,pipeDrv(); (d)stdioInit(),mathSoftInit()或 mathHardInit() (e)wdbConfig():配置并初始化目标代理机	usrConfig.c

在大多数目标板的板级支持包中，VxWorks的入口点由两个函数：romInit()和romStart()来完成，而非sysInit()。具体基于ROM的VxWorks的初始化过程如下表所示：

函数	函数功能	所在文件
1.romInit()	(a)禁止中断； (b)保存启动类型；	romInit.s

	(c)硬件初始化； (d)调用romStart();	
2.romStart()	(a)将数据段从ROM拷贝到RAM，清内存； (b)将代码段从ROM拷贝到RAM，有必要的话解压缩；	bootInit.c
3.usrInit()	(c)调用usrInit();	usrConfig.c
4.usrKernelInit()	初始化程序	usrKernel.c
5.kernelInit()	如果相应的配置文件被定义，对应函数被调用	kernelLib.c
6.usrRoot()	初始化并启动内核	usrConfig.c
7.Application routine	初始化I/O系统，驱动器，创建设备 应用程序代码	Application source file

1.1.VxWorks系统任务

目标板加电启动成功后，有如下几个任务已开始运行。

- 根任务：tUsrRoot

内核首先执行根任务tUsrRoot,其入口点为文件config/all/usrConfig.c中的usrRoot()函数，它负责初始化VxWorks工具，并创建注册、异常处理、网络通信任务和tRlogind等任务。一般来说，在所有的初始化工作完成后，根任务tUsrRoot被删除。

- 注册任务：tLogTask

注册任务tLogTask被VxWorks模块用于传送不需I/O操作的系统消息。

- 异常处理任务：tExcTask

异常处理任务tExcTask有最高优先级，它负责系统的异常情况出错处理，不能被悬置、删除或是改变其优先级。

- 网络通信任务：tNetTask

网络通信任务tNetTask负责系统级任务的网络通信。

- 目标代理任务：tWdbTask

如果目标代理程序运行在任务模式，目标代理任务tWdbTask被创建，用来响应主机目标服务器的请求。

1.2.应用软件开发指导

VxWorks系统是专为嵌入式实时应用而设计的模块化的实时操作系统。对于用户来说，一个实时应用软件是由板级支持包BSP、操作系统内核及用户选用组件、中断服务程序ISR组成。操作系统为用户提供了大量的系统调用，这是用户与操作系统的接口。针对当前开发工作和实时系统的特性，在实时应用软件的编制中要注重以下问题：

2.5.1 任务划分要合理

1.功能内聚性

对于功能联系比较紧密的各工作可以作为一个任务来运行。如果都以一个个任务来进行相互之间的消息通信，影响系统效率，不如采用任务中一个个独立的模块来完成。

2.时间紧迫性

对于实时性要求比较高的任务，要以高优先级运行，以保证事件的实时响应。

3.周期执行原则

对于一个需周期性执行的工作，应作为一个任务来运行，通过定时器以一定时间间隔激活任务。

2.5.2 防止死锁、饥饿和优先级翻转

死锁是指多个任务因为等待进入对方占据的临界区而导致的不可自行恢复的运行终止。在程序设计是要注意对死锁的预防，一个是尽量使互斥资源在相同优先级任务中使用，必须在不同优先级任务中使用时，要注意对死锁的解锁处理。

饥饿是指优先级较低的任务长期得不到系统资源（主要是指CPU资源）而造成的任务长期得无法运行。造成饥饿的主要原因是优先级较高的任务调度过于频繁或占用时间太长。合理的分配任务的优先级和对较高优先级任务的合理调度是解决饥饿的不二法门。

任务的优先级翻转是实时多任务操作系统的热门话题，它是指高优先级任务因等待低优先级任务占用的互斥资源而被较低优先级（高于低优先级但低于高优先级）的任务不断抢占的情况。有些实时多任务操作系统自身提供保护机制可对优先级翻转进行预防。在操作系统未提供保护的情况下，就需要编程人员在编程的时候注意避免优先级翻转的情况发生（如在同一优先级内使用互斥资源），或采取相应的手段进行处理（如动态的进行优先级提升）。

注：VRTX和VxWorks提供自身的防止优先级翻转机制，pSOS未提供保护机制。

2.5.3 函数的可重入性（Reentrancy）正确运用

在一个多任务环境中，函数的可重入性是十分重要的。可重入函数是一个可以被多个任务调用的过程，任务在调用时不必担心数据是否会出错。在写函数时只要考虑到尽量用局部变量（例如寄存器、堆栈中的变量），对于要使用的全局变量要加以保护（例如采用关中断、信号量等），这样构成的函数就一定是一个可重入的函数。

此外，编译器是否有可重入函数的库，与它所服务的操作系统有关，例如DOS下的Borland C和Microsoft C/C++等就不具备可重入的函数库，这是因为DOS是一个单用户单任务的操作系统。为了确保每一个任务控制自己的私有变量，在一个可重入的C函数中，须将这样的变量声名为局部变量。C编译器将这样的变量存放在调用栈上或寄存器里。

在VxWorks中，多个任务可调用同一子函数或函数库。VxWorks系统动态连接工具使这相当容易，这种共享代码让系统更加高效，易于维护。

VxWorks系统主要采用如下的几种可重入技术：

- 动态堆栈变量

许多子函数只是纯代码，除了动态堆栈变量外没有其他数据。调用程序的参数作为子函数的数据。这种子函数是完全可重入的，多个任务同时使用这种子函数，不会互相影响，因为它们各有自己的堆栈空间。

- 受保护的全局和静态变量

一些函数库包含公有数据，多个任务的同时调用很可能会导致对公有数据的破坏，使用起来要格外小心。系统采用信号量互斥机制来防止任务同时运行代码的临界区。

- 任务变量

一些公用函数要求对于每一调用程序都有明确的全局或静态变量值。为了满足这一点，VxWorks提供的任务变量允许4字节变量加入到任务上下文中，当任务切换时变量的值也切换。

编写可重入的函数，必须遵循以下的规则：

1．将所有的局部变量申明为auto（缺省态）或寄存器型。

2．尽量不要使用static或extern变量。如有必要，要用互斥机制进行保护。

2.5.4 使用名称访问资源

通过任务名、消息队列名、信号量名来调用这些资源，能保证应用系统运行的可靠性，同时也便于程序的阅读。例如系统调用taskName(),taskNameToId(),taskIsSelf()等，都方便了用户对资源的管理，更加直观化。

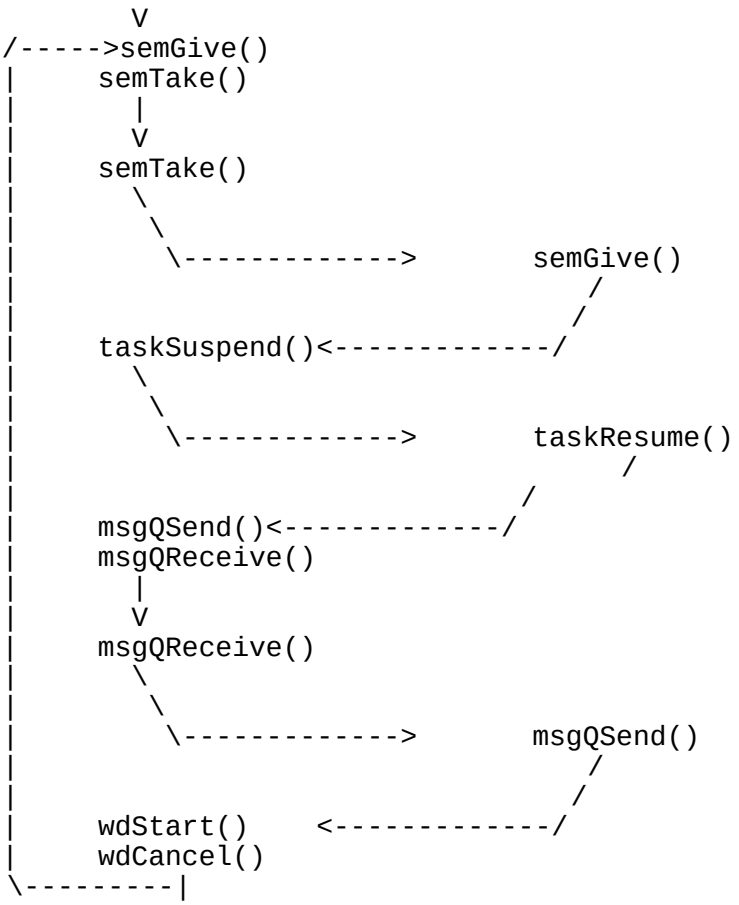
2.5.5 用户任务优先级确定

VxWorks系统中优先级分为256级，从0到255，其中0为最高优先级，255为最低优先级.任务的优先级在任务创建时被分配,但在任务运行时可通过系统调用taskPrioritySet()动态改变其优先级。当操作系统在目标板上启动成功后，系统级任务已在运行，对主机与目标机之间的通信进行管理，因此用户任务优先级要低于系统级任务，一般最高为150。同时，对于用户各任务优先级的确定，如何让各任务间良好的协同工作，有待于用户根据任务的紧急程度以及实际情况进行给定，调测过程中的摸索总结也很重要。

1.1.应用示例分析

下面通过对一具体实例的分析，对任务的创建、任务间通信、内存分配、消息管理等VxWorks系统应用更进一步的了解。(示例选自demo例子程序windDemo.c)

```
/* windDemo - repeatedly test various kernel function */

/*
modification history
--------------------
02c,23aug93,jcf  fixed synchronization.
02b,01aug93,dvs  fixed loop count printing.
02a,18mar93,dvs  took out timer/benchmark information.
                 ansified code.
              general cleanup of code to use as MicroWorks demo.
01a,12nov90,shl  written.
*/

/*
DESCRIPTION
This program repeatedly exercises different kernel facilities of
the Wind kernel.

The functions involved include the use of semaphores as sychronization
and mutual exclusion primitives, the use of taskSuspend()/taskResume() for
task control, the use of message queues for communication and the
use of watchdogs for task timeouts.

To exercise these kernel facilities two tasks are used, a high priority
task and a low priority task. The high priority task executes functions
with which the resources are not available. As the high priority task
blocks, the low priority task takes over and makes available the resources
that the high priority task is waiting for. This may sound simple at first
but the underlying execution of this test program involves context
switching, rescheduling of tasks, and shuffling of the ready queue, pend
queue, and the timer queue.

These functions are chosen because they are the most commonly used
functions in sychronization, mutual exclusion, task control, inter-task
communication and timer facilities. These are the basic building blocks of
the operating system itself and also used in applications.  Repeatedly
execution of this "death loop" is a good indication of how the system will
perform in real-life as these functions are utiltized heavily in every
application.

The following is the thread of execution of this test program.


            Higher Priority         Lower Priority
               Task1                    Task2
            ===============          ==============
               |
```




```
*/

#include "vxWorks.h"
#include "semLib.h"
#include "taskLib.h"
#include "msgQLib.h"
#include "wdLib.h"
#include "logLib.h"
#include "tickLib.h"
#include "sysLib.h"
#include "stdio.h"


/* defines */

#if FALSE
#define STATUS_INFO               /* define to allow printf() calls */
#endif

#define MAX_MSG        1          /* max number of messages in queue */
#define MSG_SIZE sizeof (MY_MSG)  /* size of message */
#define DELAY          100        /* 100 ticks */
```

```
#define    HIGH_PRI    150         /* priority of high priority task */
#define LOW_PRI        200         /* priority of low priority task */

#define TASK_HIGHPRI_TEXT    "Hello from the 'high priority' task"
#define TASK_LOWPRI_TEXT     "Hello from the 'low priority' task"


/* typedefs */

typedef struct my_msg
      {
      int    childLoopCount;       /* loop count in task sending msg */
      char * buffer;               /* message text */
      } MY_MSG;


/* globals */

SEM_ID           semId;                 /* semaphore ID */
MSG_Q_ID    msgQId;                /* message queue ID */
WDOG_ID          wdId;              /* watchdog ID */
int         highPriId;        /* task ID of high priority task */
int         lowPriId;         /* task ID of low priority task */
int         windDemoId;       /* task ID of windDemo task */


/* forward declarations */

LOCAL void taskHighPri (int iteration);
LOCAL void taskLowPri (int iteration);



/*************************************************************************
*****
*
*
* windDemo - parent task to spawn children
*
* This task calls taskHighPri() and taskLowPri() to do the
* actual operations of the test and suspends itself.
* Task is resumed by the low priority task.
*
*/

void windDemo
(
  int iteration              /* number of iterations of child code */
)
{
    int loopCount = 0;      /* number of times through windDemo */

#ifdef     STATUS_INFO
    printf ("Entering windDemo\n");
#endif /* STATUS_INFO */

    if (iteration == 0)                /* set default to 10,000 */
```

```
      iteration = 10000;

    /* create objects used by the child tasks */

    msgQId  = msgQCreate (MAX_MSG, MSG_SIZE, MSG_Q_FIFO);
    semId   = semBCreate (SEM_Q_PRIORITY, SEM_FULL);
    wdId    = wdCreate ();

    windDemoId = taskIdSelf ();

    FOREVER                  //while(1)
    {

      /* spawn child tasks to exercise kernel routines */

      highPriId = taskSpawn ("tHighPri", HIGH_PRI, VX_SUPERVISOR_MODE,
            1000,(FUNCPTR) taskHighPri, iteration,0,0,0,0,0,0,0,0,0);

      lowPriId = taskSpawn ("tLowPri", LOW_PRI, VX_SUPERVISOR_MODE,
            1000,(FUNCPTR) taskLowPri, iteration,0,0,0,0,0,0,0,0,0);


      taskSuspend (0);       /* to be waken up by taskLowPri */

#ifdef     STATUS_INFO
      printf ("\nParent windDemo has just completed loop number %d\n",
            loopCount);
#endif /* STATUS_INFO */

      loopCount++;
    }
 }

/*************************************************************************
*****
*
*
* taskHighPri - high priority task
*
* This tasks exercises various kernel functions. It will block if the
* resource is not available and relingish the CPU to the next ready task.
*
*/

LOCAL void taskHighPri
(
  int iteration             /* number of iterations through loop */
)
{
    int    ix;                    /* loop counter */
    MY_MSG msg;                   /* message to send */
    MY_MSG newMsg;                /* message to receive */

    for (ix = 0; ix < iteration; ix++)
      {

      /* take and give a semaphore - no context switch involved */
```

```
      semGive (semId);
      semTake (semId, 100);        /* semTake with timeout */


      /*
       * take semaphore - context switch will occur since semaphore
       * is unavailable
       */

      semTake (semId, WAIT_FOREVER);    /* semaphore not available */

      taskSuspend (0);       /* suspend itself */


      /* build message and send it */

      msg.childLoopCount = ix;
      msg.buffer = TASK_HIGHPRI_TEXT;

      msgQSend (msgQId, (char *) &msg, MSG_SIZE, 0, MSG_PRI_NORMAL);

/*
 * read message that this task just sent and print it - no context
 * switch will occur since there is a message already in the queue
*/

      msgQReceive (msgQId, (char *) &newMsg, MSG_SIZE, NO_WAIT);

#ifdef     STATUS_INFO
      printf ("%s\n Number of iterations is %d\n",
            newMsg.buffer, newMsg.childLoopCount);
#endif /* STATUS_INFO */

/*
 * block on message queue waiting for message from low priority task
 * context switch will occur since there is no message in the queue
 * when message is received, print it
 */

      msgQReceive (msgQId, (char *) &newMsg, MSG_SIZE, WAIT_FOREVER);

#ifdef     STATUS_INFO
      printf ("%s\n Number of iterations by this task is: %d\n",
            newMsg.buffer, newMsg.childLoopCount);
#endif /* STATUS_INFO */

      /* test watchdog timer */

  wdStart (wdId, DELAY, (FUNCPTR) tickGet, 1);

  wdCancel (wdId);
 }
}

/*************************************************************************
*****
```

```
*
*
* taskLowPri - low priority task
*
* This task runs at a lower priority and is designed to make available
* the resouces that the high priority task is waiting for and
*subsequently unblock the high priority task.
*
*/

LOCAL void taskLowPri
(
  int iteration             /* number of times through loop */
)
{
    int    ix;                    /* loop counter */
    MY_MSG msg;                   /* message to send */

    for (ix = 0; ix < iteration; ix++)
      {
      semGive (semId);       /* unblock tHighPri */

      taskResume (highPriId);           /* unblock tHighPri */

      /* build message and send it */

      msg.childLoopCount = ix;
      msg.buffer = TASK_LOWPRI_TEXT;
      msgQSend (msgQId, (char *) &msg, MSG_SIZE, 0, MSG_PRI_NORMAL);
      taskDelay (60);
      }

    taskResume (windDemoId);      /* wake up the windDemo task */
    }
```

附：常用系统调用

1.taskSpawn　　创建(产生并激活）新任务

```
  int   taskSpawn
   (
    char    *name,      /*新任务名称*/
    int     priority,   /*任务的优先级*/
    int     options,    /*任务可选项*/
    int     stackSize,  /*任务堆栈大小*/
    FUNCPTR entryPt,    /*任务入口函数*/
    int     arg1,       /*任务入口函数所带参数1~10*/
    int     arg2, int   arg3,
    int     arg4, int   arg5,
    int     arg6, int   arg7,
```

```
    int     arg8, int   arg9,
    int     arg10
   )
```

函数运行成功返回任务ID号，否则为ERROR。

任务可选项的几种模式如下表：

名 称	值	描 述
VX_FP_TASK	0x8	运行带浮点的协处理器
VX_NO_STACK_FILL	0x100	不使用0xee填充堆栈
VX_PRIVATE_ENV	0x80	用私有环境运行任务
VX_UNBREAKABLE	0x2	断点失效
VX_SUPERVISOR_MODE		用户任务常用值

2.taskDelete 删除一个任务

```
 STATUS     taskDelete
 (
  int   tid   /*删除任务的ID号*/
 )
```

删除指定ID号的任务，并释放任务所占有的内存

3.taskDelay 延迟任务

```
 STATUS     taskDelay
 (
  int   ticks /*延迟的时间滴答数*/
 )
```

任务延迟为某一任务休眠一定时间提供了简单的处理方法，一般用于任务的周期性循环执行。当输入参数为NO_WAIT（其值为零）时，表示将所延迟的任务切换到同一优先级就绪队列的尾部。

4.taskSuspend 任务悬置

```
  STATUS    taskSuspend
    (
      int   tid   /*被悬置的任务ID号*/
     )
```

5.taskResume 恢复任务

```
  STATUS    taskResume
    (
      int   tid   /*恢复的任务ID号*/
```

```
    )
```

6.msgQCreate　　创建并初始化消息队列

```
  #include <msgQLib.h>
  MSG_Q_ID  msgQCreate
   (
      int   maxMsgs,          /*队列所能容纳的最大消息数目*/
      int   maxMsgLength,     /*每一消息的最大长度*/
      int   options           /*消息入列方式*/
   )
```

消息入列方式有两种：MSG_Q_FIFO 先进先出，按时间先后顺序考虑；MSG_Q_PRIORITY 按消息优先级考虑。

7.msgQSend　　向一消息队列发送消息包

```
   STATUS   msgQSend
    (
      MSG_Q_ID    msgQId,     /*所发向的消息队列名*/
      char  *     buffer,     /*消息包所在缓冲区指针*/
      UINT        nBytes,     /*消息包长度*/
      int         timeout,    /*等待的时间长度*/
      int         priority    /*优先级*/
   )
```

该函数将长度为nBytes的缓冲区buffer消息包发向消息队列msgQId. 如果任务正在等待接收该消息队列的消息包，消息将立即被送到第一个等待的任务。如果没有任务等待此消息，消息包被保留在消息队列中。

参数timeout指明：当消息队列已满时，等待消息队列有空间时所等待的时间。超过该时间还没空间可用的话，消息包被舍弃。它有两个特殊值：NO_WAIT（0）立即返回，不管消息包是否被发送；WAIT_FOREVER（-1）一直等待消息队列有空间可用。

参数priority指明发送的消息的优先级，可能值有：MSG_PRI_NORMAL(0)正常优先级，将消息置于消息队列的尾部；MSG_PRI_URGENT(1)紧急消息，将消息置于消息队列的首部。

8.msgQReceive　　接收消息

```
  int msgQReceive
   (
      MSG_Q_ID    msgQId,     /*接收消息的消息队列ID号*/
      char  *     buffer,     /*接收消息的缓冲区指针*/
```

```
    UINT        maxNBytes,  /*缓冲区长度*/
    int         timeout     /*等待时间*/
)
```

该函数从消息队列msgQId接收消息，将其拷贝到最大长度为maxNBytes的缓冲区buffer。如果消息包长度超过maxNBytes,多余部分被舍弃。等待时间timeout有两个特殊值： NO_WAIT（0）立即返回；WAIT_FOREVER（-1）一直等待消息队列有消息可取。

9.msgQDelete　　删除一个消息队列

```
STATUS   msgQDelete
 (
   MSG_Q_ID    msgQId      /*要删除的消息队列ID号*/
 )
```

任何因发送或接收该消息队列的消息的任务都将解阻，并返回错误errno.

10. wdCreate　　创建看门狗定时器

```
WDOG_ID   wdCreate(void)
```

11. wdStart　　启动定时器

```
STATUS   wdStart
  (
   WDOG_ID     wdId,       /*看门狗定时器ID号*/
   int         delay,      /*延迟值，以滴答计*/
   FUNCPTR     pRoutine,   /*超时函数*/
   int         parameter   /*超时函数的参数*/
  )
```

12. wdCancel　　取消一个当前工作的定时器

```
STATUS   wdCancel
 (
   WDOG_ID   wdId  /*被取消的定时器ID号*/
 )
```

该函数只是让定时器的延迟值为零来取消其工作。

13. wdDelete　　删除定时器

```
STATUS   wdDelete
 (
   WDOG_ID   wdId  /*被删除的定时器ID号*/
 )
```

14. semBCreate　　创建并初始化二进制信号量

```
SEM_ID   semBCreate
 (
   int             options,          /*信号量选项*/
   SEM_B_STATE     initialState      /*信号量初始化状态值*/
 )
```

信号量初始化状态值有两种：SEM_FULL（1）或SEM_EMPTY（0）。选项参数指明被阻塞任务的入列方式：基于优先级（SEM_Q_PRIORITY）和先进先出（SEM_Q_FIFO).

15. semCCreate　创建并初始化计数信号量

```
SEM_ID   semCCreate
 (
   int       options,          /*信号量选项*/
   int       initialCount      /*信号量初始化计数值*/
 )
```

选项参数指明被阻塞任务的入列方式：基于优先级（SEM_Q_PRIORITY）和先进先出（SEM_Q_FIFO).

16. semGive　给出信号量

```
STATUS   semGive
 (
  SEM_ID     semId  /*所给出的信号量ID号*/
 )
```

17. semTake　获得信号量

```
STATUS   semTake
 (
  SEM_ID     semId  /*所要得到的信号量ID号*/
  int        timeout /*等待时间*/
 )
```

如果任务在规定时间内未得到信号量，函数semTake返回错误。等待时间值WAIT_FOREVER和NO_WAIT分别表示一直等待和不等待。

18. semDelete　删除信号量

```
STATUS   semDelete
 (
  SEM_ID     semId  /*要删除的信号量ID号*/
 )
```

该函数释放与此信号量相关的资源，所有等待此信号量的任务解阻。