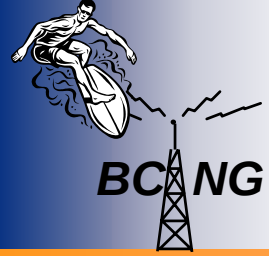
面向二十一世纪的嵌入式系统设计技术

第七讲：
VxWorks实时操作系统
RTOS(三)：VxWorks

任课教员：徐　欣　博士
主讲教员：张志群　博士

国 防 科 大 电 子 科 学 与 工 程 学 院
嵌 入 式 系 统 开 放 研 究 小 组



基于VxWorks的嵌入式开发技术

张 志 群

2002年12月

Email: zzqun73@263.net

内　容

- 网络产品实现方法的演进
- 实时系统
- VxWorks介绍
- Tornado开发环境介绍
- BSP
- 设备驱动

实现方法的演进

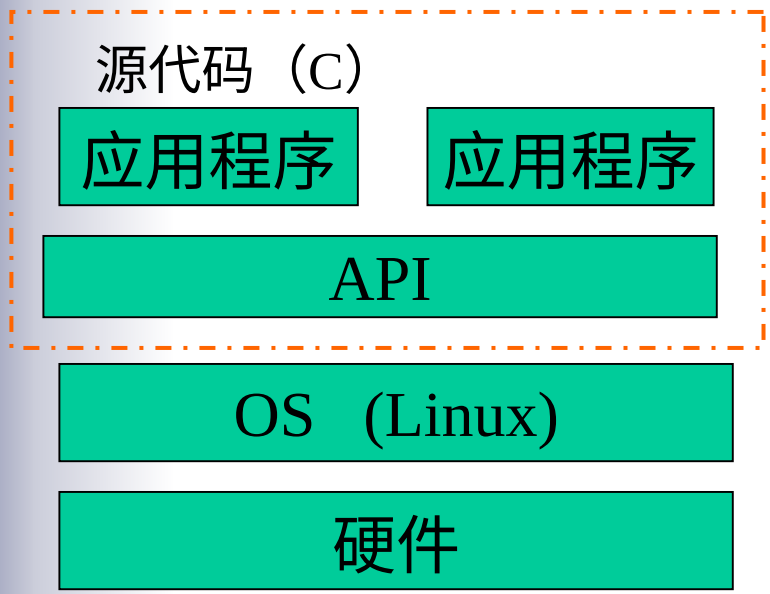
- 基于PC 的架构 (80’s-90’s Early)

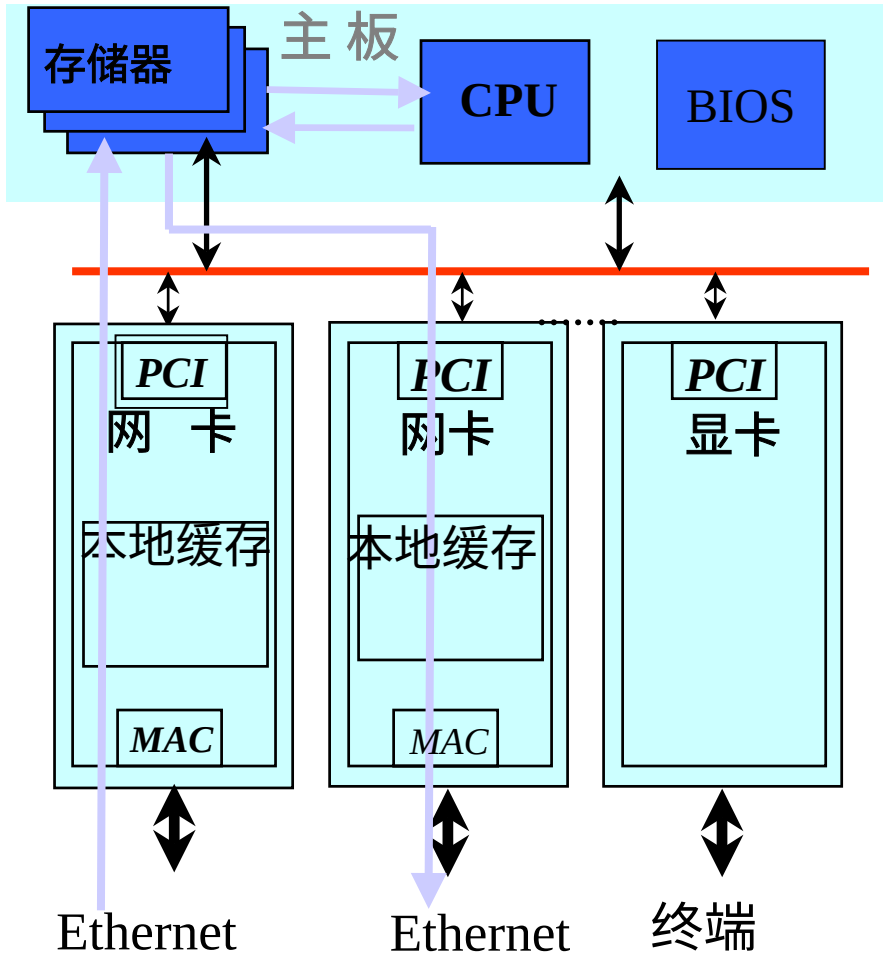
  软件实现

- 基于ASIC+MPU的架构 (90’s Middle)

  软件控制，硬件线速处理

基于PC的架构





问题：

1、接口速率问题（up to 100M）；

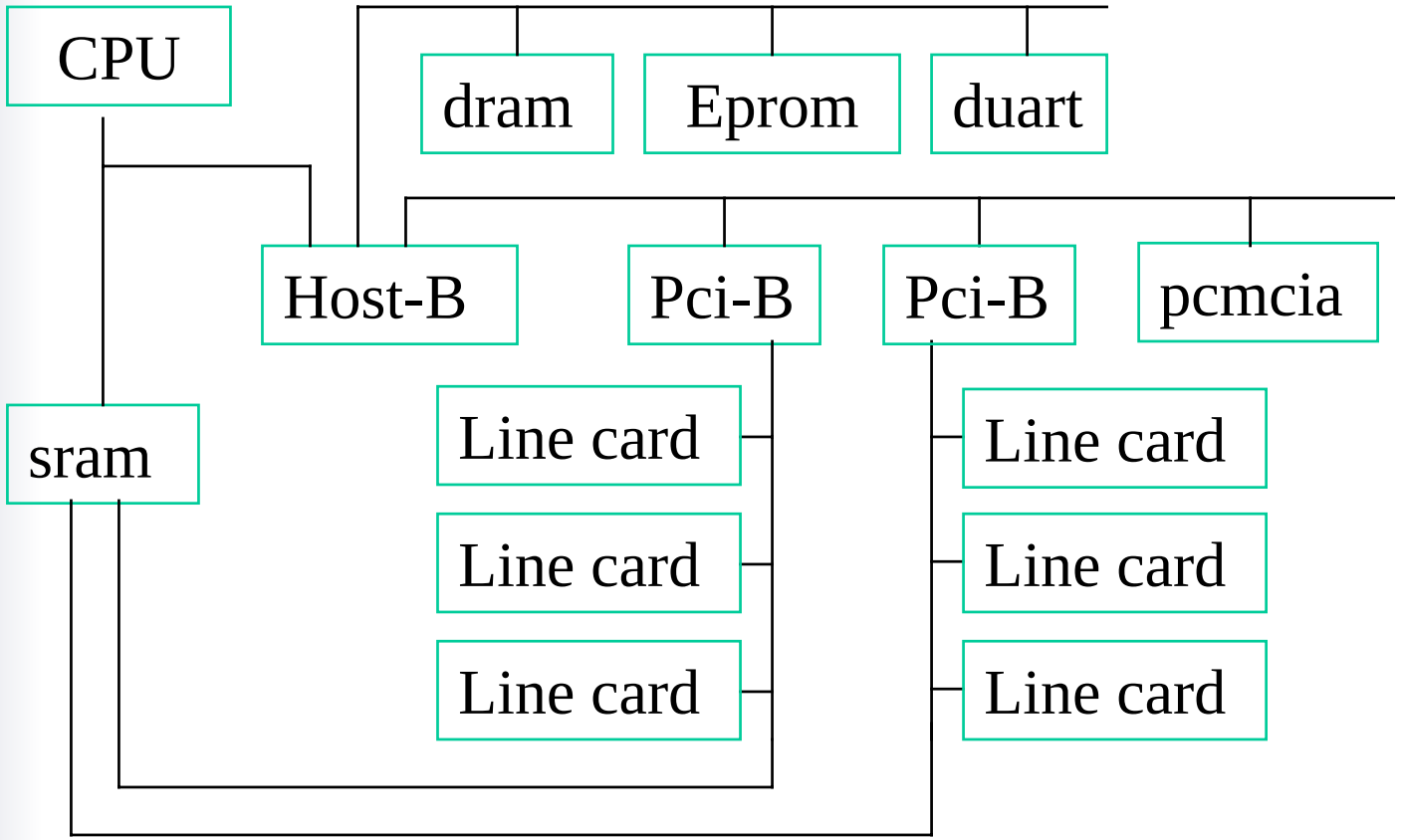
2、处理容量问题；

3、安全性问题；

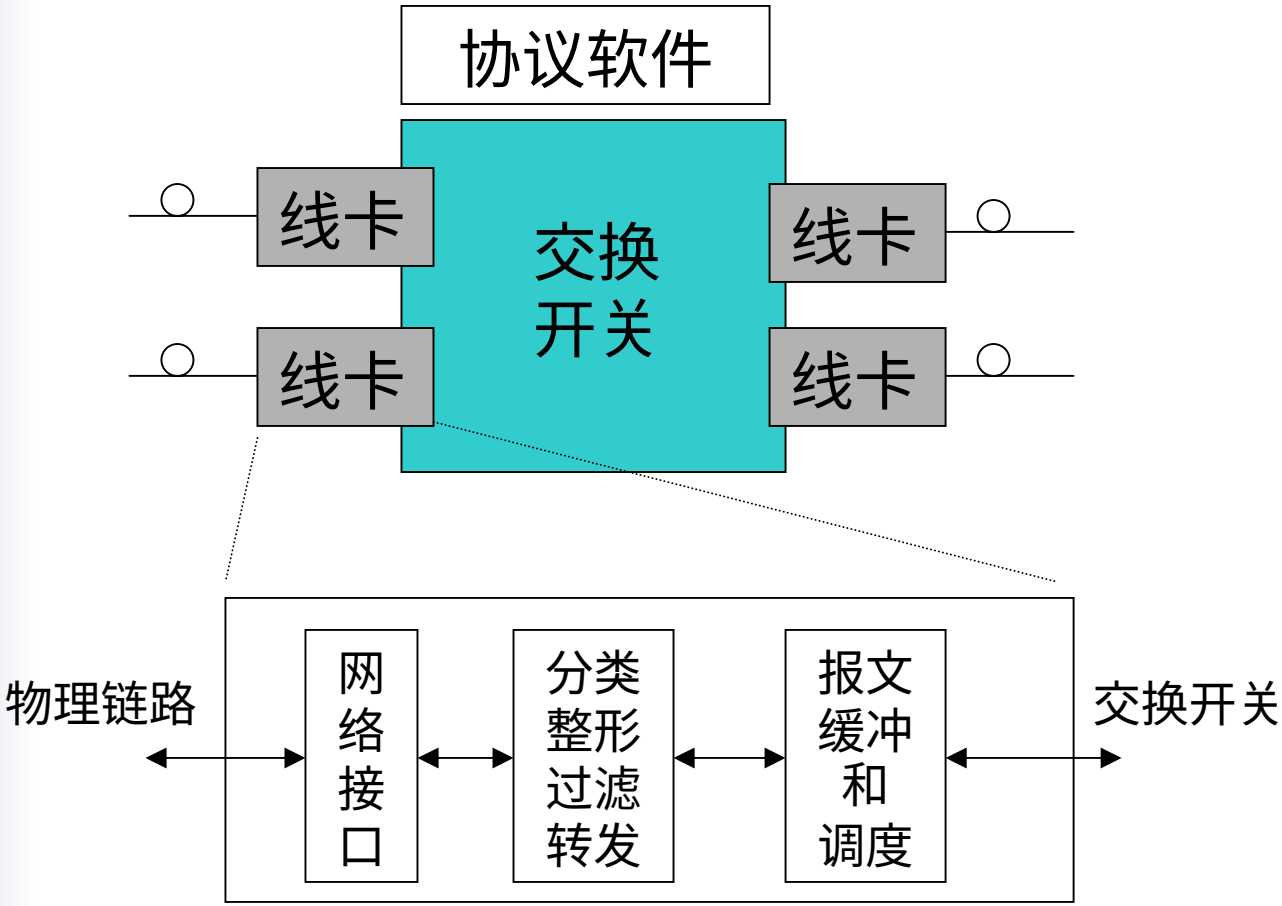
上层软件

工作组/园区路由器的基本结构

-Cisco 7200 router



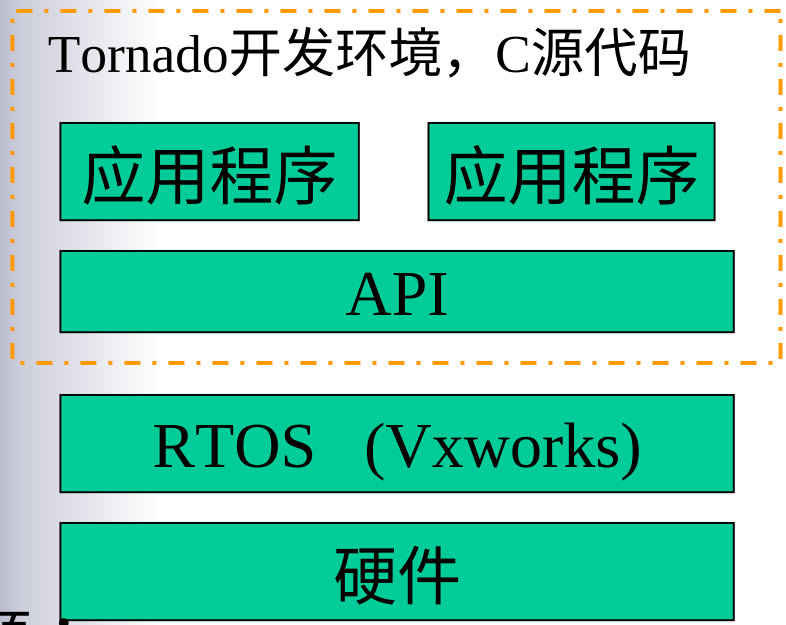
路由器基本结构

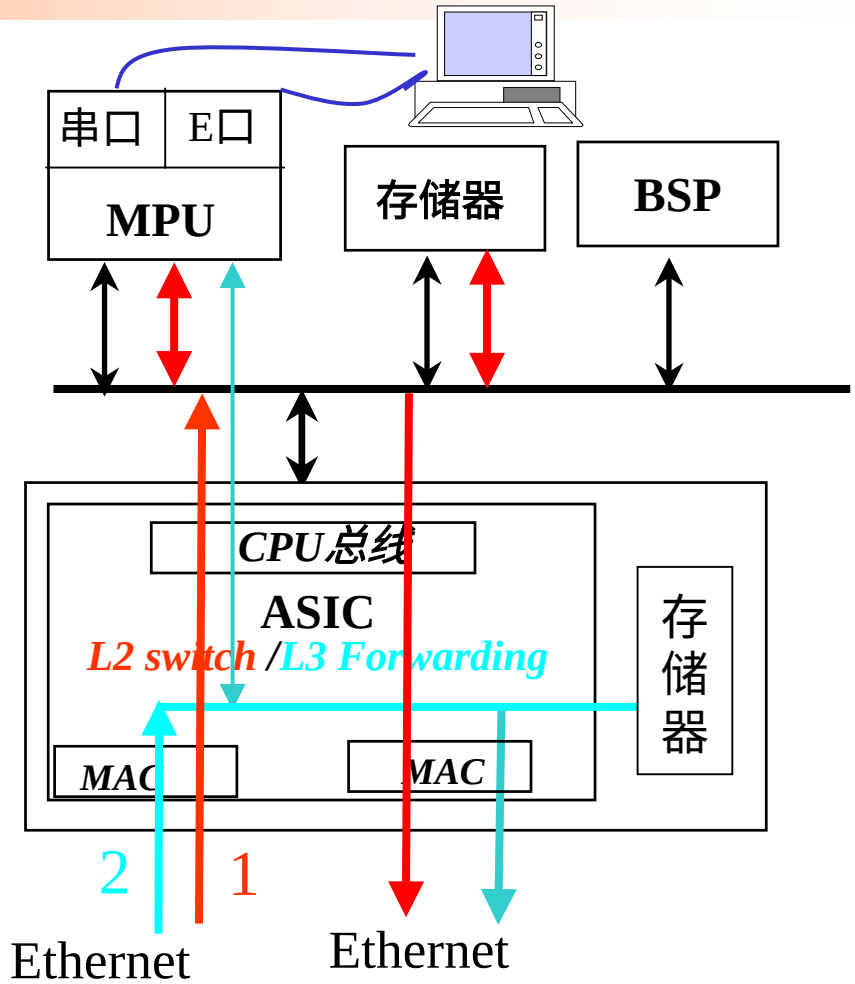


基于ASIC+MPU的架构（1）

路由器层次结构（硬件＋上层软件）

基于ASIC+MPU的架构（2）





问题：

1、速率与功能受限于Chip；

2、ASIC，速率低；FPGA，周期长；

4、可扩展性差。

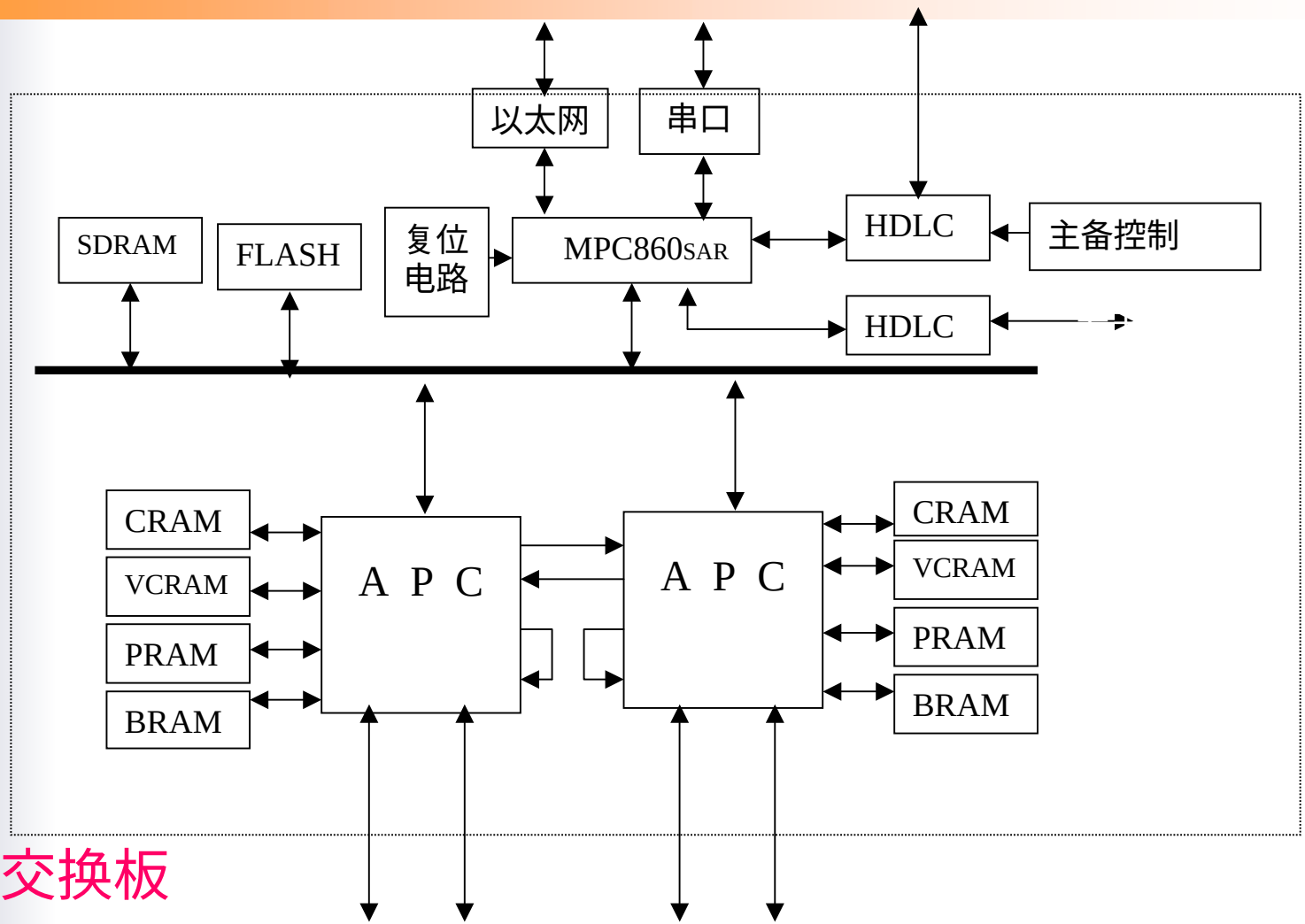
ATM层次模型

AAL	管理平面		维护网络、连接路由、执行操作 //网络层
	控制平面 信令 消息	高层	建立呼叫和连接 //网络层
		CPCS+SSCS	信令实体间的连接 //链路层
		SAR	分段重组 //链路层
	用户平面 用户 信息	高层	类似 OSI 高层的功能 //会话层
		CPCS+SSCS	端到端无差错连接 //传输层
		SAR	分段重组 //传输层
ATM			相当于链路层下边界
PHY			相当于物理层

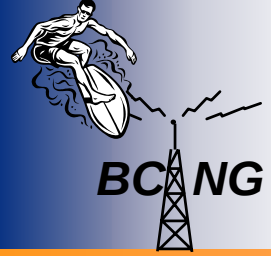
注：AAL层对用户信息相当L4的下边界，对控制信息相当L2的下边界


AIAS 2000
ATM 综合接入交换机
AIAS 2000
ATM 综合接入交换机

单板介绍


以太网
串口
SDRAM
FLASH
复位
电路
MPC860SAR
HDLC
主备控制
HDLC
CRAM
VCRAM
PRAM
BRAM
A P C
A P C
CRAM
VCRAM
PRAM
BRAM

交换板



内　容

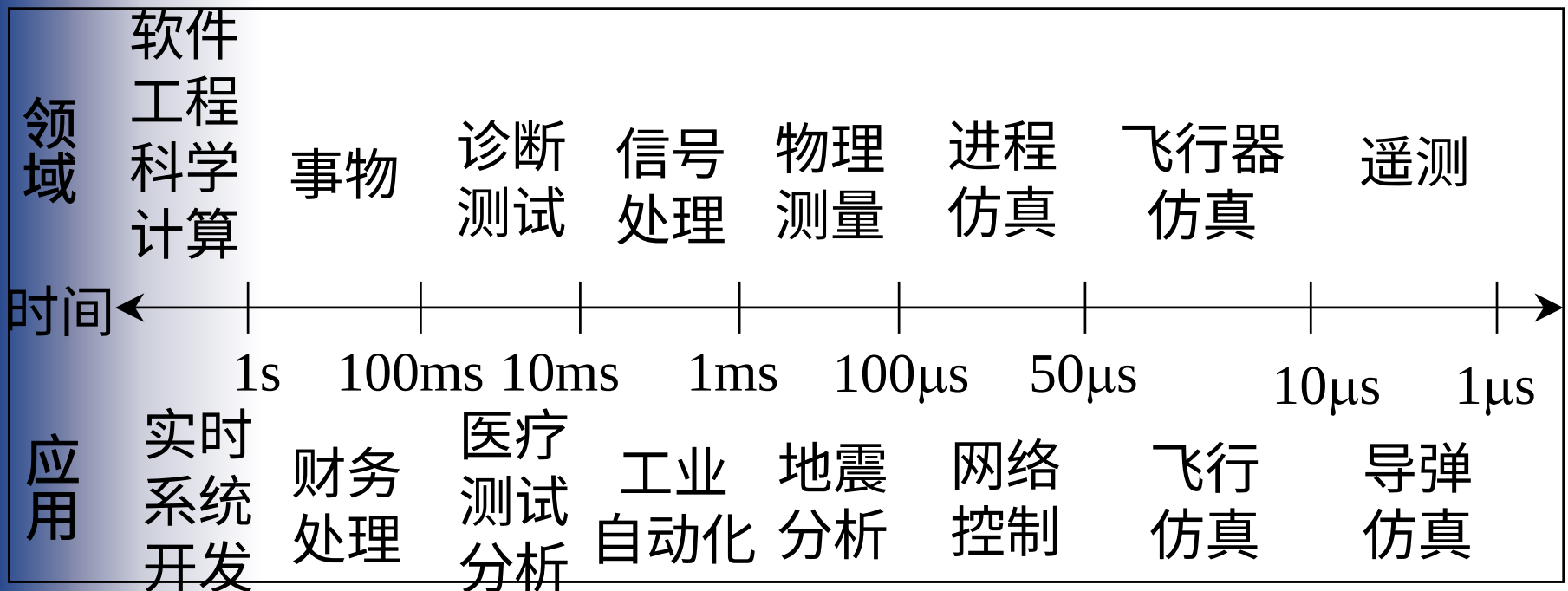
- 网络产品实现方法的演进
- 实时系统
- VxWorks介绍
- Tornado开发环境介绍
- BSP
- 设备驱动

实时系统定义

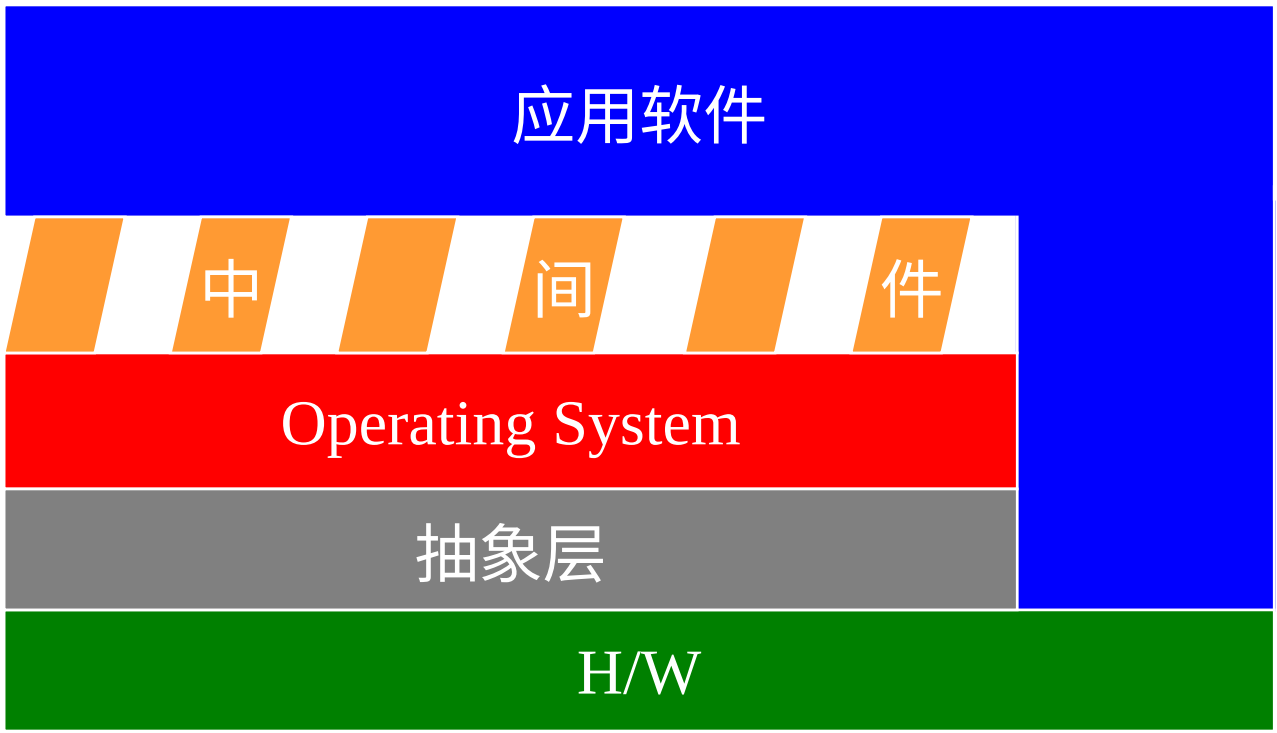
- 对于给定一个时间约束量 ε > 0，如果系统S在T1时刻接受到输入，在T2时刻给出合理的输出，且使T2-T1< ε 。则称系统S满足要求的时间 ε 的实时性，通常称系统S为实时系统。
- “ 正确、但迟后的结果也是错误的 ”

实时系统

- 时间约束是相对的
  - 关键因素：系统对外部激励的响应时间



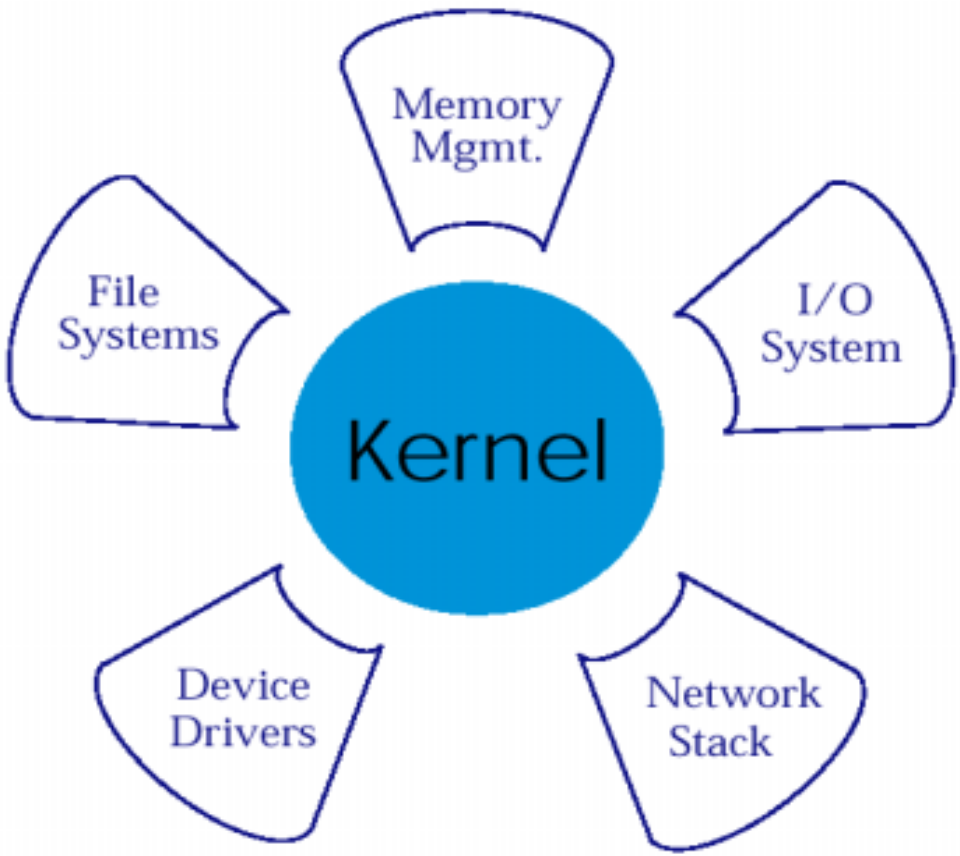
计算机系统的组成

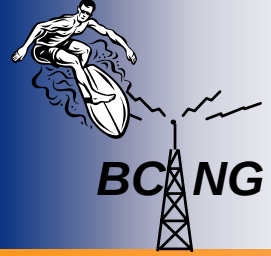


实时操作系统

- 实时操作系统允许应用程序满足严格的时间要求
- 多任务内核
  - 实时调度（基于优先级的抢占）
  - 任务间通信
  - 互斥
- 其它功能作为库由Kernel调度

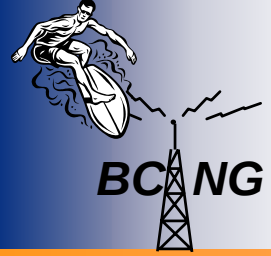
实时操作系统


Memory Mgmt.
File Systems
I/O System
Kernel
Device Drivers
Network Stack



与非实时操作系统的区别

- 基于优先级抢占的调度
- 高效、快
- 小，可配置



实时操作系统中的概念

- 多任务
- 调度
- 优先权
- 时间片
- 通信和同步
- 资源共享
- 事件
- 互斥
- 信号量
- 邮箱

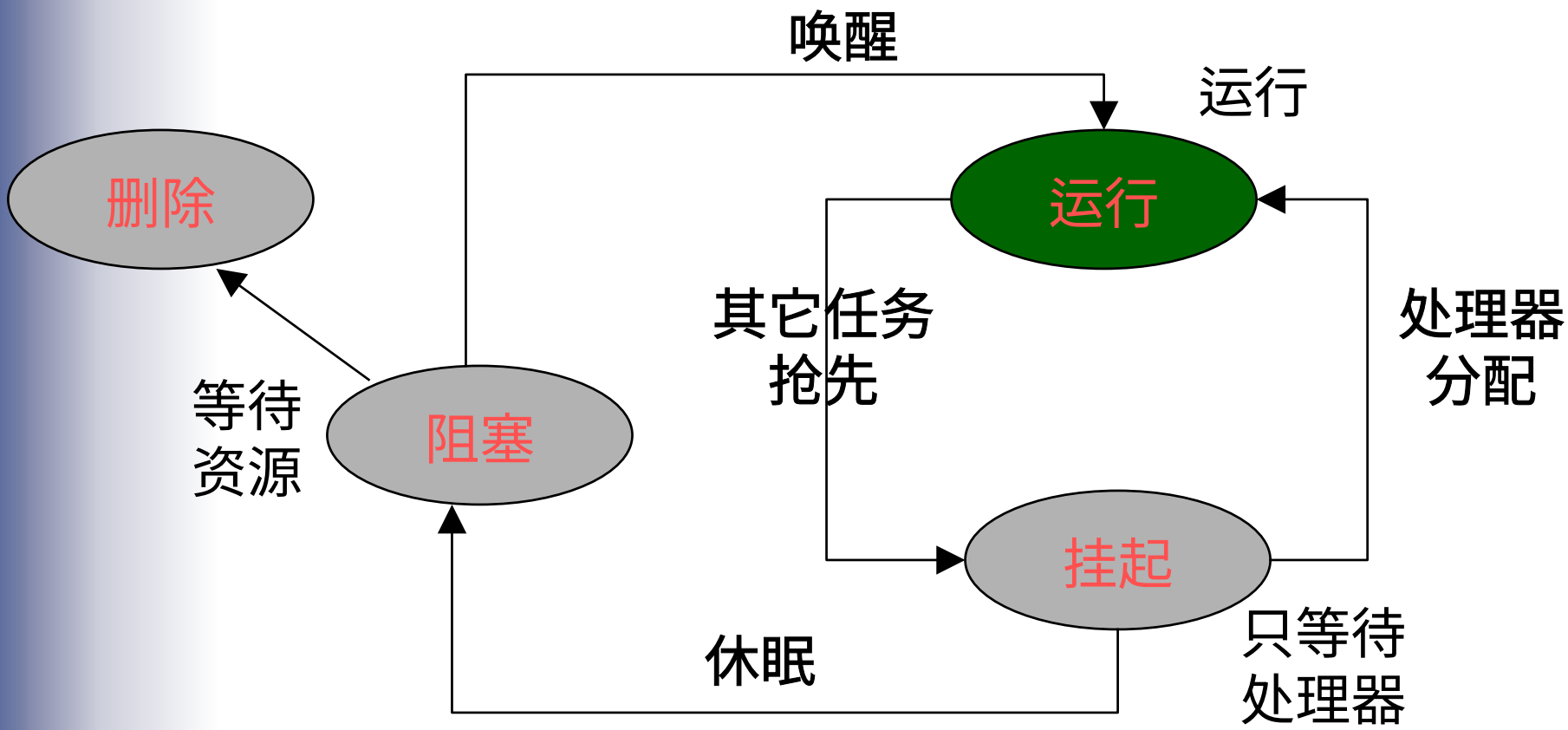
多任务

- 目的：优化系统资源（CPU 时间、内存、磁盘、驱动器...）的使用
- 多任务 OS
  - 允许多个任务并发和独立地在系统上运行
  - 实现独占和共享系统资源
  - 根据所需的输入/输出资源，控制任务执行
- 任务（Task）、Process、Agent、Thread
  - 逻辑整体动态（进程）实体

Task与Program的区别

- 程序 （program）
  - 静态实体，由一个或多个指令序列，管理一组数据（内部或外部变量）
- 任务（task）
  - 动态实体，运行一个或多个程序，以实现处理器上指定的活动

任务的不同状态



调 度

- 调度器负责任务的状态管理和当前任务的选择
- 分发器选择由调度器推举的当前的任务（有效的上下文切换）
- 处理器分配算法的判据选择有：
  - 任务优先、其寿命、消耗的CPU时间、等等
  - 用调度器动态调整

优先权

注释

唯一重要的优先权是任务之间的相对优先权

通信和同步

- 任务互操作，以便合作完成公共活动
- 目的
  - 管理任务间共享的系统资源，避免系统死锁

资源共享

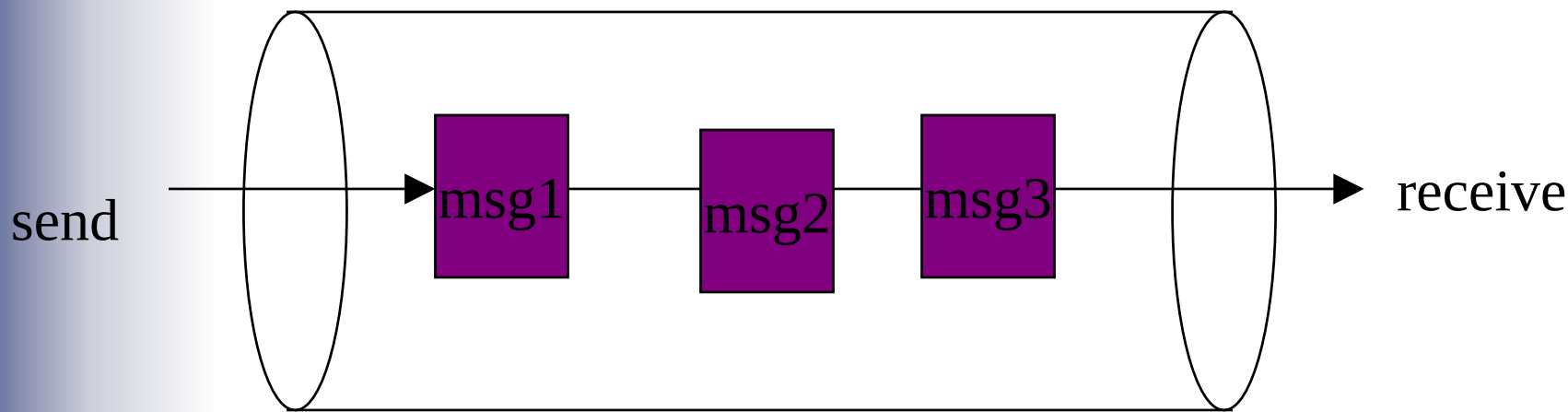
- 资源可以是
  - 软件或硬件
  - 局部（仅在一个任务内使用）或公用
  - 能共享的资源有最大的进入能力

互 斥

- 避免数据或者代码不一致的机制
- 互斥
  - 当几个任务共享非重入资源(判据)时发生互斥
- 信号量
  - 信号量类同于售票机

邮 箱

- 邮箱是任务之间的交换区
- 2个队列与邮箱有关
  - 一个是消息队列
  - 一个是任务队列（多个发送和接受）



内　容

- 网络产品实现方法的演进
- 实时系统
- VxWorks介绍
  - ✓ 基本定义
  - ✓ 内核(Wind)
  - ✓ 任务间通信
  - ✓ POSIX、ISR、Watchdog
  - ✓ I/O系统及其他
- Tornado开发环境介绍
- BSP
- 设备驱动

VxWorks

Vxworks操作系统是一个嵌入式实时操作系统（RTOS）。

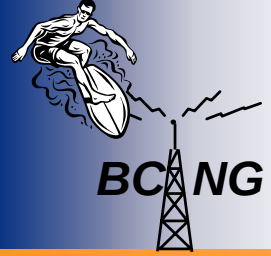
Vxworks与其它实时OS一样，基于以下两个重要机制：

* 多任务环境及任务间通信
* 硬件中断处理

Vxworks多任务内核完成的功能是：实时调度，任务间通信及互斥。
其它功能则作为系统库围绕在内核周围，它们可根据需要进行剪裁。

Vxworks与非实时系统的不同之处在于：

Vxworks的优先抢占机制基于调度，
Vxworks对外部事件的反应和处理快，
Vxworks容量小并且可配置（微内核结构）



VxWorks组件

- 高性能实时内核
- POSIX(1003.1b)兼容接口
- I/O系统
- 本地文件系统
- C/C++开发支持
- 共享内存
- 虚拟内存
- 目标机驻留工具

VxWorks组件（续）

- 工具库
- 性能评估工具
- 目标机代理
- 板支持包（BSP）
- VxWorks仿真器和逻辑分析仪（WindView）
- 网络组件

内　容

- 网络产品实现方法的演进
- 实时系统
- VxWorks介绍
  - ✓ 基本定义
  - ✓ 内核(Wind)
  - ✓ 任务间通信
  - ✓ POSIX、ISR、Watchdog
  - ✓ I/O系统及其他
- Tornado开发环境介绍
- BSP
- 设备驱动

内核（Kernel）

- VxWorks的内核叫Wind
- 包括
  - *多任务抢占和优先权调度*
  - 任务间同步和通信
  - 中断处理
  - WatchDog定时器
  - 内存管理

多任务内核、任务机制、任务间通信和中断处理机制是VxWorks运行环境的核心。其中，多任务和任务间通信是现代实时操作系统的基石。

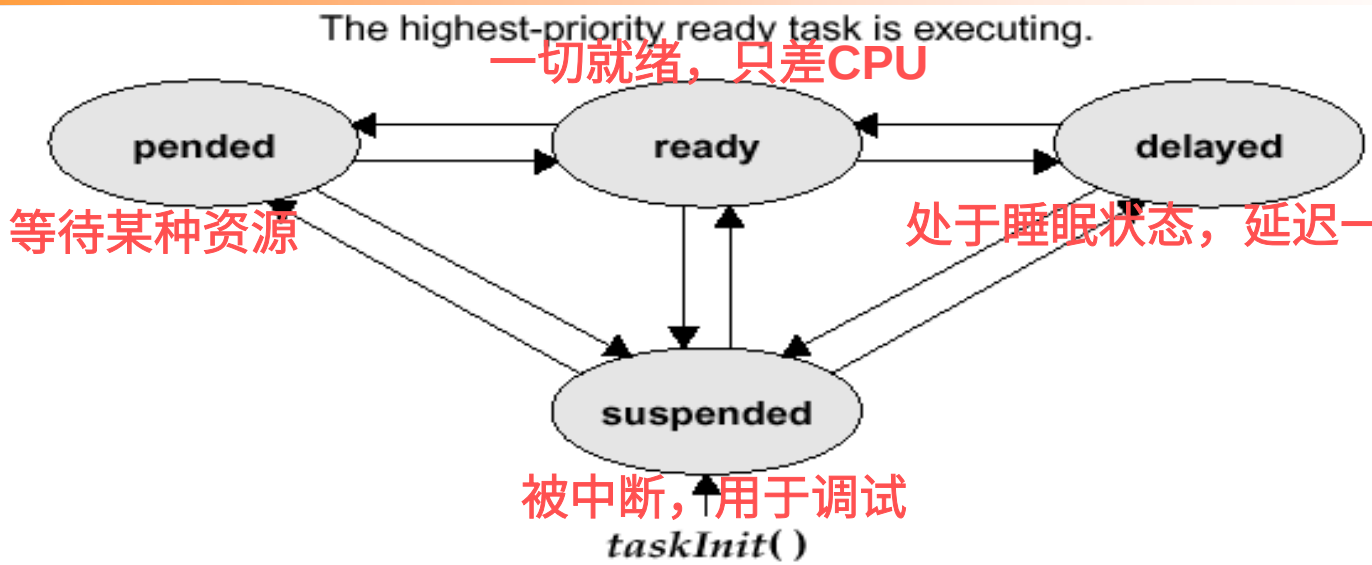
内核—多任务

- 任务控制块(TCB)
- 任务状态转换
- 任务调度
- 任务控制
- 任务扩展
- POSIX任务调度接口（略）
- 任务错误状态：errno
- 任务异常处理(Exception Handle)
- 共享代码和重入(Shared code and reentrancy)
- 系统任务(System Task)

内核—任务控制块（TCB）

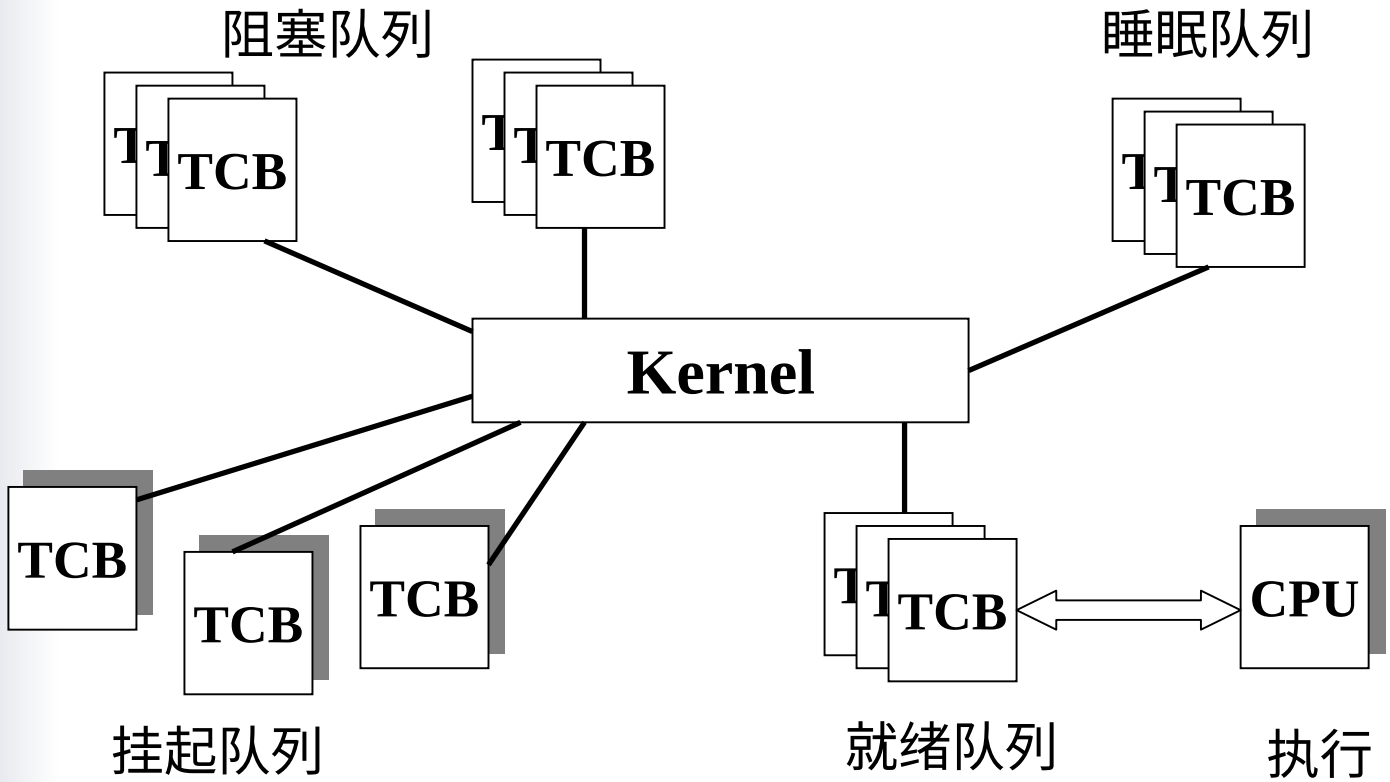
- 保存任务的上下文，一个任务的上下文包括：
  - 程序执行指针
  - CPU寄存器和浮点寄存器
  - 动态变量和函数调用的堆栈
  - 标准输入、输出和错误的I/O分配
  - 延迟定时器
  - 时间片定时器
  - 内核控制结构
  - 信号处理器
  - 调试和性能监视值

内核—任务状态转换



ready	→	pended	*semTake( ) / msgQReceive( )*
ready	→	delayed	*taskDelay( )*
ready	→	suspended	*taskSuspend( )*
pended	→	ready	*semGive( ) / msgQSend( )*
pended	→	suspended	*taskSuspend( )*
delayed	→	ready	expired delay
delayed	→	suspended	*taskSuspend( )*
suspended	→	ready	*taskResume( ) / taskActivate( )*
suspended	→	pended	*taskResume( )*
suspended	→	delayed	*taskResume( )*

任务状态队列



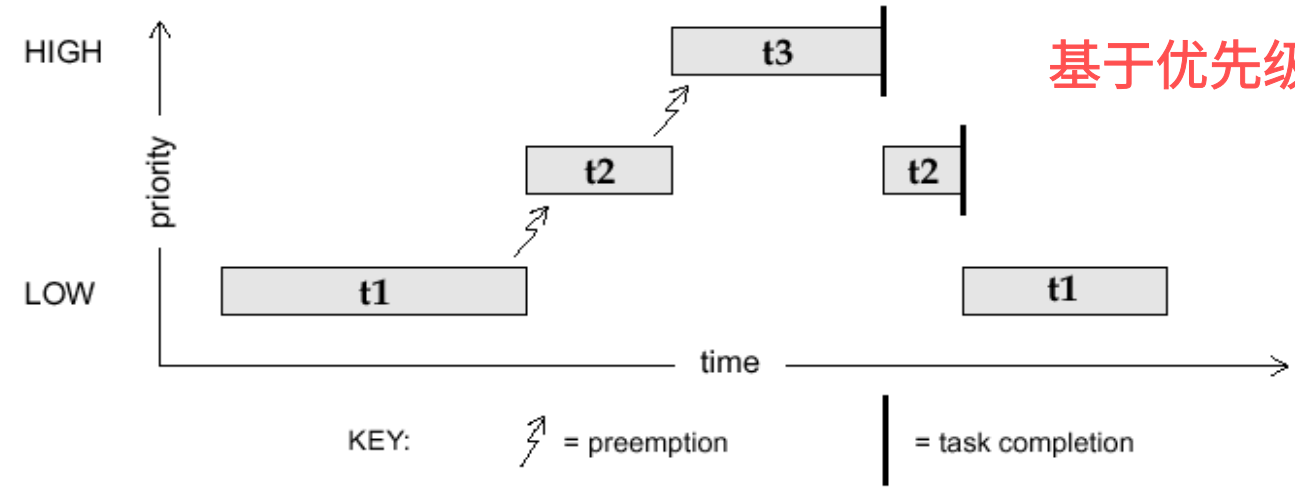
内核负责维护系统中
所有任务的当前状态。

一个任务的状态转变是
应用调用内核调用的结果。

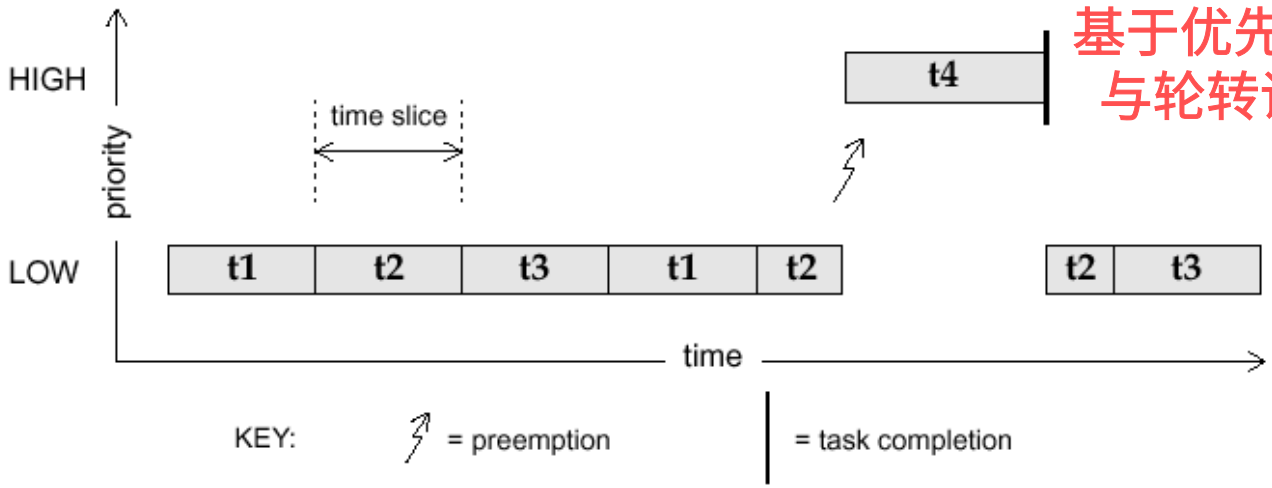
内核— Wind 任务调度

- 优先级抢占调度（缺省）
  - 优先级高的Task抢占CPU
  - 0－255，256个优先级，0级最高，255级最低
- 轮询调度
  - 优先级相同时，多个Task轮流占用CPU
- 抢占锁定
  - 独占CPU，其它高优先级的Task不能抢占CPU

内核— Wind Task调度



基于优先级的抢占调度



基于优先级的抢占调度
与轮转调度算法结合

内核— 任务控制函数

- taskSpawn()：创建并激活一个task（定位）
- taskInit()：初始化一个新task
- taskActivate()：激活一个task

低级操作

参数：
- ✓ Task ID：32bits，指向task控制块的指针(ID 0指调用task)
- ✓ Task Name：代表task的ASCII字符串

所有从目标机启动的任务以字母 t开头，所有从主机启动的任务以字母 u开头

- Task选项
- 获取Task信息
- Task删除和删除保险
- Task控制：改变task的状态

内核— 任务扩展函数

- 为扩展与task相关的功能，Wind提供勾连（hook）功能，在task创建、切换、删除时，自动唤起相关的勾连例程。
- 在TCB中有一个字段存放此扩展
- *taskCreateHookAdd*( ) 每个task创建时增加一个例程
- *taskCreateHookDelete*( )
- *taskSwitchHookAdd*( )
- *taskSwitchHookDelete*( )
- *taskDeleteHookAdd*( )
- *taskDeleteHookDelete*( )

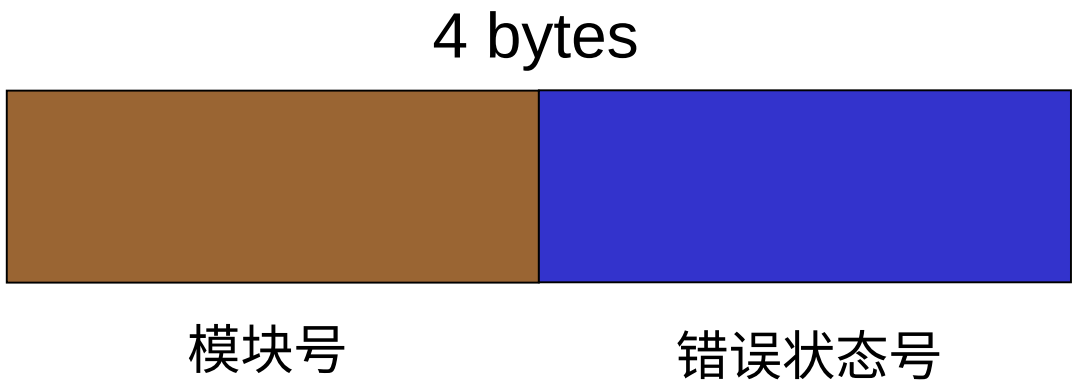
内核— 任务错误状态：errno

- Errno是一个预定义的全局变量
- 多任务情况下，每个任务有自己的errno，作为task上下文的一部分，中断服务程序（ISR）也有自己的errno
- 惯例：程序返回OK(0)表示成功，ERROR(-1)表示失败；若返回指针，则NULL(0)表示失败。
- 返回ERROR或NULL时，通常设置errno表示具体错误代码
- Errno总表示最近的错误状态，不会被清除
- 如果errno在错误状态符号表(statSymTbl)中有对应字符串，则可以使用printErrno()显示错误内容(不能用在ISR中)
- 尽量使用logMsg()来显示错误信息，因为它在task和ISR下都能工作

内核— Errno分配

- Errno编码中使用前两个字节表示产生错误的模块，后两个字节表示每个错误号
- VxWorks系统的模块号为1-500，0用于源代码兼容
- 应用程序的模块号为大于500的正数和所有负数

errno



内核—用户自定义errno

- 在用户头文件目录下创建xxModNum.h，定义自己的模块：

```
#define M_lemLib    (512 << 16)
```

- 在用户头文件目录下创建lemErr.h，定义错误状态号的宏：

```
#include "xxModNum.h"
/* lemLib errors */
#define S_lemLib_LEM_INIT_FAIL          (M_lemLib | 1)
#define S_lemLib_LEM_CLOSE_FAIL         (M_lemLib | 2)
#define S_lemLib_MSG_TYPE_ERROR         (M_lemLib | 3)
```

- 重新编译系统错误表statTbl.o

  makeStatTbl *systemHeaderDir userHeaderDir*

  编译得到的statTbl.c文件，产生statTbl.o

内核—用户自定义errno

- 在VxWorks中包含组件 *development tool components* > *symbol table components* > *error status table*.
- 重新编译VxWorks
- 如果要将错误代码加入WindShell，需要将新模块的错误字符串加入文件host/resource/tcl/errnoTbl.tcl或者$(HOME)/.wind/windsh.tcl中

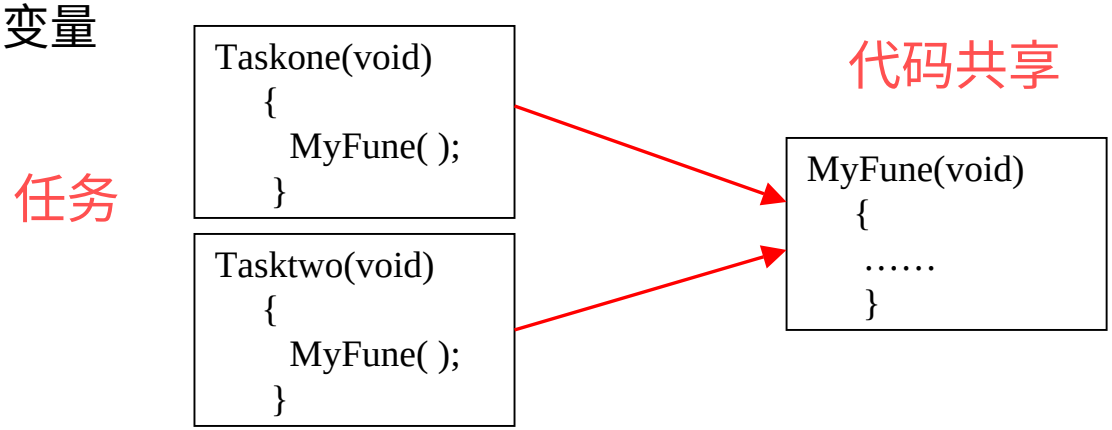
```
set M_lemLib            [expr 512 << 16]
set errnoTbl [expr $M_lemLib | 1] S_lemLib_LEM_INIT_FAIL
set errnoTbl [expr $ M_lemLib | 2] S_lemLib_LEM_CLOSE_FAIL
set errnoTbl [expr $ M_lemLib | 3] S_lemLib_MSG_TYPE_ERROR
```

内核— 任务异常处理

- VxWorks的异常处理包负责所有异常处理
- 缺省的异常处理是挂起此task，并保存异常点的task状态，其它task继续运行
- Tornado的开发工具可以查看挂起task的状态
- 也可以通过信号(signal)将某硬件或软件的异常处理与自己的处理程序关联

内核—共享代码和重入

- 被多个task调用的代码叫共享代码，共享代码必须可重入
- 若代码要修改全局/静态变量，则不可重入，否则会引起数据混乱
- 多数例程可重入，但如果例程foo() 还有一个对应的foo_r()例程，则foo()是不可重入的
- VxWorks使用以下可重入技术：
  - 动态堆栈变量
  - 全局变量和静态变量由信号量守护
  - 任务变量



内核— VxWorks系统任务

- Root Task，tUsrRoot是内核执行的第一个task，它创建其它task，完成任务后被停止并删除
- 日志Task，tLogTask记录系统日志消息，而不必执行I/O
- 异常Task，tExcTask支持VxWorks的异常处理
- 网络Task，tNetTask处理网络的任务级别的功能
- 目标机代理Task，tWdbTask处理debug请求
- Task可选组件
  - tShell、tRlogind、tTelnetd、tPortmapd

内　容

- 网络产品实现方法的演进
- 实时系统
- VxWorks介绍
  - ✓ 基本定义
  - ✓ 内核(Wind)
  - ✓ 任务间通信
  - ✓ POSIX、ISR、Watchdog
  - ✓ I/O系统及其他
- Tornado开发环境介绍
- BSP
- 设备驱动

任务间通信

Vxworks提供了一套丰富的任务间通信机制，包括：

- 共享数据结构/内存共享（Shared memory）：
  简单的数据共享方法
- 信号量(Semaphore)： 用于基本的互斥及同步
- 消息队列(Message Queue)和管道(Pipes)：
  用于同一CPU上任务间消息的传递
- 套接口（Socket）和远程程序调用（RPC）：
  用于网络上任务间的通信
- 信号(Signals)： 用于异常处理

任务间通信——共享数据结构

- 共享数据结构
- VxWorks中所有task存在于一个线性的地址空间中，所以task之间共享数据结构很容易
- 这些数据结构可以是
  - 全局变量
  - 线形缓冲区
  - 环形缓冲区
  - 链表
  - 指针

任务间通信——共享内存

Vxworks提供了三种共享内存的对象（shared-memory objects）来实现在不同任务之间和不同CPU的任务间的高速同步和通信。

- **共享信号量（shared semaphores）：**

  有二进制，记数型两种，用于在不同CPU上的任务间的同步，和对共享数据结构的互斥访问

- **共享消息队列（shared message queues）：**

  允许多个处理器上的任务交换消息

- **共享内存分区（shared-memory partitions）：**

  有系统类型和用户类型可以用于为不同处理器上的任务分配公共数据空间

任务间通信——互斥

- 为避免内存访问竞争，需要内存访问互锁
- 有许多方法可以实现资源的访问互斥：
  - 禁止中断、禁止抢占、信号量
- 禁止中断(最强大，时间要尽量短)
  - int lock = intLock();
  - . . *critical region that cannot be interrupted* .
  - intUnlock (lock);
- 禁止抢占(可以被中断)
  - taskLock ();
  - . . *critical region that cannot be interrupted* .
  - taskUnlock ();

任务间通信——信号量

- 信号量(Semaphore)是解决互斥和任务同步的最主要手段
- 信号量提供比中断禁止和抢占禁止更精细的互斥
- 信号量协调任务的执行和外部事件，以实现任务同步
- Wind将信号量优化为三类，以解决不同类型的问题：
  - 二进制：最快，最通用的信号量，用于同步和互斥
  - 互斥：专门解决互斥问题而优化的二进制信号量：优先权继承、删除保护和递归
  - 计数：类似二进制，但记录信号量发放的次数，为守护资源的多个实例而优化
- POSIX信号量(略)

任务间通信——消息队列

- 任何task或者ISR都可以向消息队列中放入消息，多个task可以向同一个消息队列中发送消息或者从其中接收消息
- 两个task之间的双向通信，通常需要两个消息队列，每个方向一个
- 消息队列中的消息个数和每个消息的长度都是可变的
- VxWorks支持两种消息队列库：
  - Wind Queue
  - POSIX Queue
- 超时
- 优先级

任务间通信——管道（Pipes）

- 管道是虚拟的I/O设备
- Task使用标准的I/O例程操作管道
  - Open, read, write, ioctl
- 管道支持select函数

任务间通信—— Task网络通信

- Sockets
  - 支持TCP/UDP
  - 与BSD 4.4 UNIX兼容
- 远端过程调用（RPC）
  - RPC允许一个机器上的进程调用同一个机器或者别的机器上运行的过程
  - RPC内部使用socket作为底层通信手段

任务间通信——信号(Signal)

- 信号可以异步地改变task的执行流程
- 任何task或ISR都可以向某个task发送一个信号
- 收到信号的task立即挂起，下次被调度时运行信号处理例程
- 信号处理例程使用接收task的上下文和堆栈
- 信号处理例程应当作ISR来对待
- 即使接收task被阻塞，依然能够唤醒信号处理例程
- 信号更适用于错误或异常处理，而不是一般的Task通信
- Wind支持两种信号接口：
  - UNIX BSD风格、POSIX兼容

内　容

- 网络产品实现方法的演进
- 实时系统
- VxWorks介绍
  - ✓ 基本定义
  - ✓ 内核(Wind)
  - ✓ 任务间通信
  - ✓ POSIX、ISR、Watchdog
  - ✓ I/O系统及其他
- Tornado开发环境介绍
- BSP
- 设备驱动

POSIX兼容

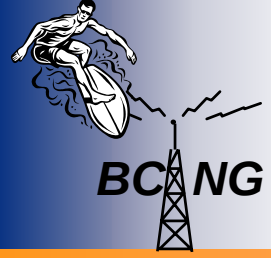
- POSIX：the Portable Operation System Interface
- ISO/IEEE制定的一组接口，以支持应用程序在不同操作系统上的源代码的移植，使用这些接口有助于将软件从一个操作系统移植到另一个操作系统。
- 实时操作系统对应的POSIX接口标准为1003.1b（原来的1003.4），VxWorks几乎都支持，重要包括：
  - 异步I/O
  - 信号量
  - 消息队列
  - 内存管理
  - 排队信号
  - 调度
  - 时钟和定时器

POSIX接口

- VxWorks Wind内核包括:

  POSIX接口和专为VxWorks设计的接口，对应两种不同调度。

- POSIX调度
  - ✓ 基于进程（不可直接访问内存，父子关系）
  - ✓ 基于FIFO，优先数越高，优先级越高
- Wind调度
  - ✓ 基于任务（可直接访问内存）
  - ✓ 基于优先级的抢占式调度，优先数越低，优先级越高
- POSIX时钟和定时器，支持多个虚拟时钟
- POSIX内存上锁接口，支持分页和交换技术



中断服务代码(ISR)

- 为尽快响应中断请求，中断服务例程在task上下文以外的特别的上下文中运行，因而唤醒ISR，不需要切换任务上下文
- 使用intConnect()，可以将C函数与任何中断连接起来
- 有些结构支持独立的中断堆栈，有些不支持，由BSP决定
- ISR的特别限制(没有TCB)：
  - ISR不能调用可能使自己被阻塞的例程
  - ISR不能分配和释放内存，或调用包含分配和释放内存的函数
  - ISR不能通过I/O访问设备，因为可能阻塞
  - ISR使用logMsg()来向控制台(console)打印消息
  - ISR不能使用浮点协处理器
  - ISR可以使用所有VxWorks的工具库，支持errno

中断服务代码

- ISR出现异常时无法挂起，VxWorks将异常描述存放到低端内存，然后重新启动系统；VxWorks boot ROM测试低端内存，并将异常描述打印到控制台上
- 可以为某些事件预留最高级别的中断，实现0时延响应
- ISR到Task的通信
  - 共享内存和环形缓冲区
  - 信号量（ISR只能释放信号量）
  - 消息队列（使用NO_WAIT参数）
  - 管道（只能write）
  - 信号（ISR用信号通知Task）

WatchDog定时器

- VxWorks提供Watchdog Timer机制，允许任何C函数与一个特定的时间延迟相联系。通常，作为系统中断服务程序的一部分来维护。
- 利用Watchdog来处理任务时限
  - *wdCreate*( ) 分配并初始化一个watchdog定时器
  - *wdDelete*( ) 终止并删除一个watchdog定时器
  - *wdStart*( ) 启动一个watchdog定时器
  - *wdCancel*( ) 取消一个正在计时的watchdog

内　容

- 网络产品实现方法的演进
- 实时系统
- VxWorks介绍
  - ✓ 基本定义
  - ✓ 内核(Wind)
  - ✓ 任务间通信
  - ✓ POSIX、ISR、Watchdog
  - ✓ I/O系统及其他
- Tornado开发环境介绍
- BSP
- 设备驱动

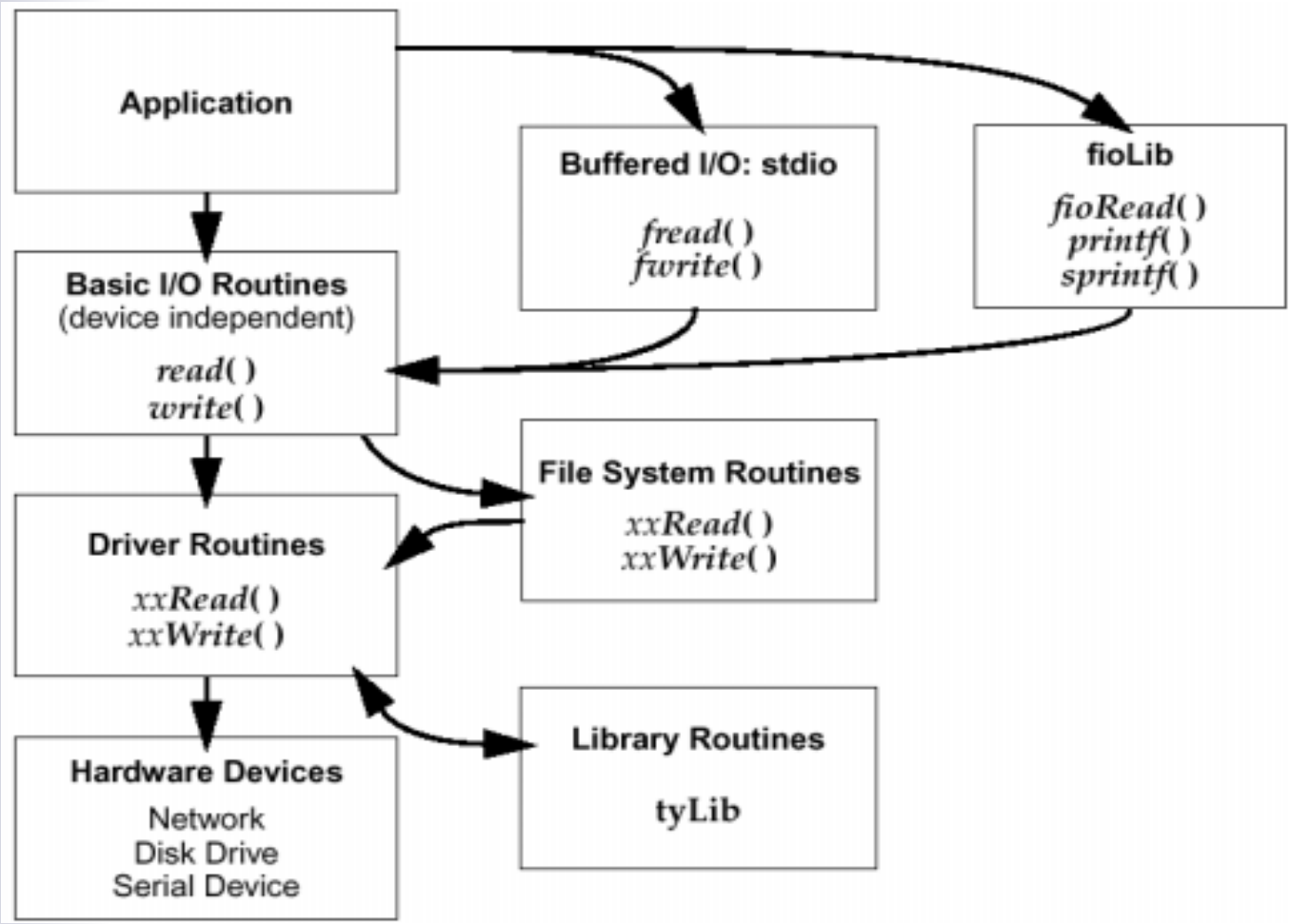
I/O系统

- VxWorks的I/O系统提供对多种设备的统一访问
  - 基本I/O例程：***creat( )***, ***remove( )***, ***open( )***, ***close( )***, ***read( )***, ***write( )***, ***ioctl( )***.
  - 高级I/O例程：***printf( )***, ***scanf( )***
- VxWorks提供标准缓冲I/O例程
  - ***fopen( )***, ***fclose( )***, ***fread( )***, ***fwrite( )***, ***getc( )***, ***putc( )***
- VxWorks的I/O系统还提供POSIX兼容的异步I/O
  - 在执行任务其它动作的同时执行输入和输出
- VxWorks可动态安装和卸载设备驱动而无需重新启动系统

VxWorks的I/O系统

- VxWorks的I/O系统为各种设备提供简单的、统一的、与设备无关的接口，包括：
  - 面向字符的设备，如终端
  - 随机访问块设备，如磁盘
  - 虚拟设备，如task间的管道和socket
  - 监视和控制设备，如数字/模拟I/O设备
  - 访问远端设备的网络设备
- VxWorks为基本I/O和有缓冲区I/O提供标准C库
- 基本I/O库与UNIX兼容；有缓冲区I/O库与ANSI C兼容
- VxWorks的I/O系统设计使之比大多数I/O系统更快更灵活，这对实时系统很重要

VxWorks的I/O系统



文件、设备及驱动

- 在VxWorks中，应用程序通过打开文件来访问I/O设备
- 文件指下面两种事物之一：
  - 未结构化的“原始”设备，如串行通信通道或者任务间管道
  - 位于一个结构化的、随机访问的、包含文件系统的设备上的逻辑文件
- I/O设备有两个级别：基本I/O和有缓冲区I/O
- 文件名和缺省设备：
  - /usr　　NFS网络设备
  - Host:　　Non-NFS网络设备
  - Dev:　　dosFs文件系统设备

基本I/O

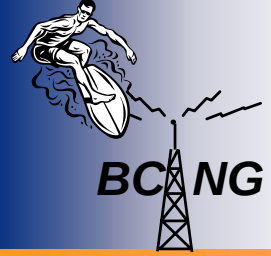
- 基本I/O在VxWorks中是最低级的I/O，其接口与标准C库的I/O原语兼容
  - *creat*( ) 创建一个文件
  - *remove*( ) 删除一个文件
  - *open*( ) 打开一个文件(也可创建一个文件)
  - *close*( ) 关闭一个文件
  - *read*( ) 读一个已经创建或打开的文件
  - *write*( ) 写一个已经创建或打开的文件
  - ftruncate () 将一个文件切割成指定大小
  - *ioctl*( ) 对文件或设备执行特定控制功能
- 在基本I/O级别，文件用文件描述字(fd)指示，它是一个整数，在creat()或open()时返回
- Fd不用后应及时关闭，以节约资源

标准输入、标准输出和标准错误

- 下面的文件描述字（fd）保留作特殊用途：
  - 0 = 标准输入
  - 1 = 标准输出
  - 2 = 标准错误输出
- 它们不会由create( )返回，但可以重定向到其它fd
- 系统缺省情况下将标准fd重定向到控制台，任务缺省情况下使用全局重定向的定义
- 它们可以被全局重定向：
  - **ioGlobalStdSet (stdFd, fileFd);**
- 每个任务的重定向将覆盖全局定义
  - ioTaskStdSet (0, stdFd, fileFd);

ioctl( )

- 为了运行某些不适合于其它基本I/O的I/O函数，使用ioctl( )设置或查询一些属性，譬如：
  - 当前设备可输入多少字节
  - 设置设备的特定选项
  - 获取某个文件系统的信息
  - 。。。
- Ioctl( )的参数是fd、指示请求函数的代码、和一个函数需要的参数：
  - *result* = ioctl (*fd*, *function*, *arg*);
- 将一个tty设备的数据波特率设为9600的例子：
  - *status* = ioctl (*fd*, FIOBAUDRATE, 9600);



等待在多个fd上的select( )

- 将任务挂起于多个文件描述字或者超时后返回
- 与Unix和Windows兼容
- selectLib提供两种支持
  - 任务级别支持：任务等待多个设备激活
  - 设备驱动支持：等待设备I/O的同时检测挂起的任务

有缓存的I/O：Stdio

- 低级I/O调用会带来一些开销，为提高I/O访问的灵活性和效率VxWorks提供了有缓存的I/O机制
- Stdio的I/O函数提供透明的缓存机制，提高访问效率
  - **FILE *fp;**
  - ***fp* = fopen ("/usr/foo", "r");**
- 标准输入、输出、和出错
  - Stdin
  - Stdout
  - Stderr

其它格式化I/O

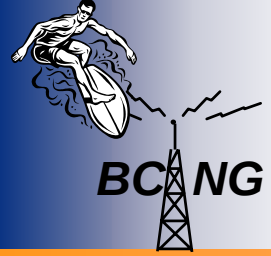
- 特殊情况:
  - ***printf*( ), *sprintf*( ), and *sscanf*( )属于fioLib，没有缓存**
- 附加函数*:*
  - ***printErr( ) and fdprintf( )***
- 消息日志
  - 提供日志功能
  - 避免当前任务访问I/O
  - 可以重定向到其它I/O

异步输入输出

- 异步输入输出（AIO）：让一般内部处理和I/O操作同步进行
- 提高任务的效率
- AIO的例程（见参考手册）
- AIO控制块
- AIO的使用

本地文件系统

- VxWorks包括多种使用块设备（磁盘）的本地文件系统
- VxWorks的I/O结构使VxWorks可以同时有多种文件系统：
  - MS-DOS兼容文件系统：dosFs
    - 功能强大
  - RT-11兼容文件系统：rt11Fs
    - 无分层文件组织结构，文件连续
  - 原始磁盘文件系统：rawFs
    - 整个磁盘作为一个文件
  - SCSI序列文件系统：tapeFs
    - 磁带的一卷作为一个文件
  - CD-ROM文件系统：cdromFs
    - ISO 9660



虚拟内存

- 虚拟内存支持有内存管理单元（MMU）的目标板

内　容

- 网络产品实现方法的演进
- 实时系统
- VxWorks介绍
- Tornado开发环境介绍
- BSP
- 设备驱动

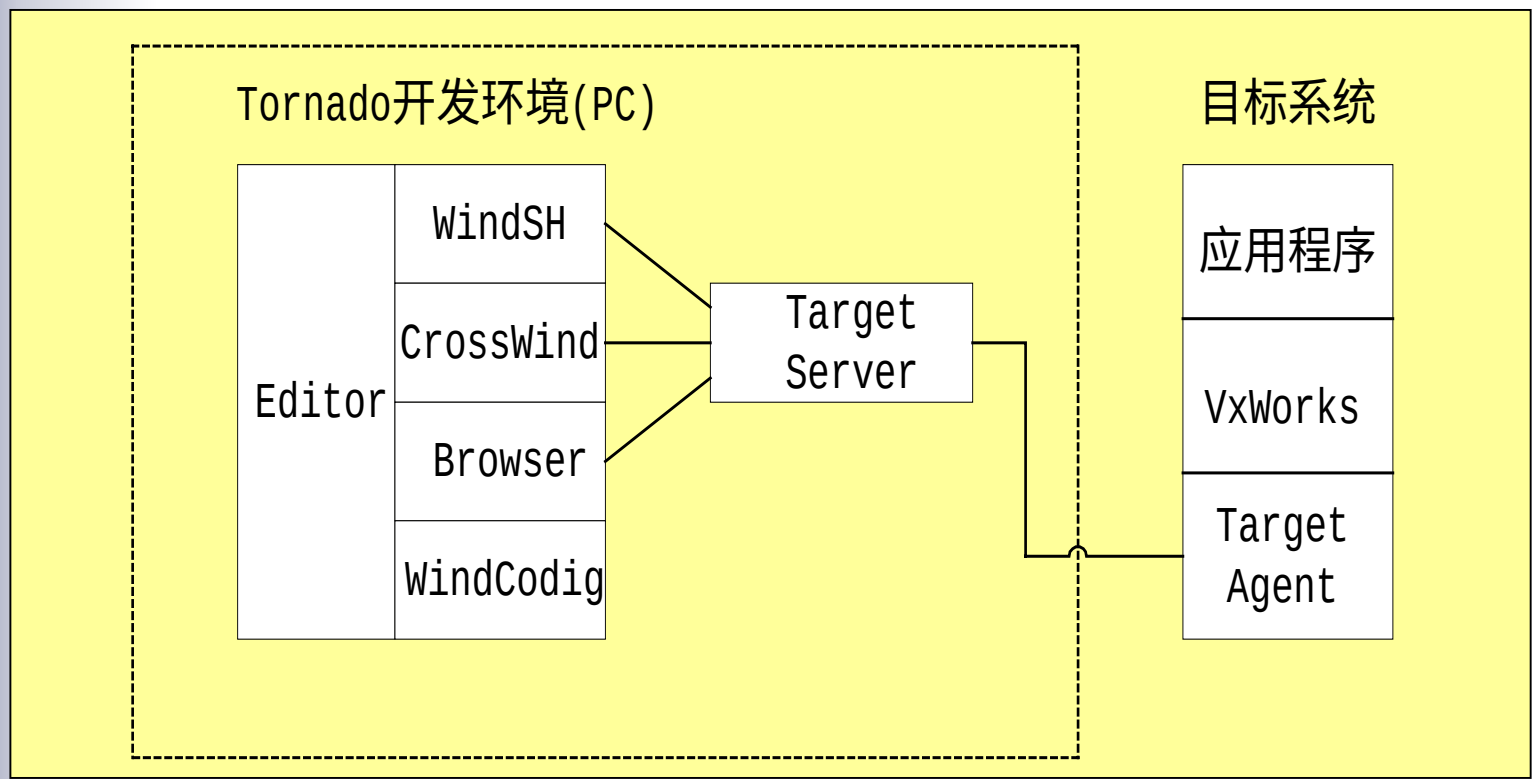
Tornado （1）

Tornado提供了Vxworks系统开发的集成开发环境，其中Tornado源代码编辑器包括下列特性：

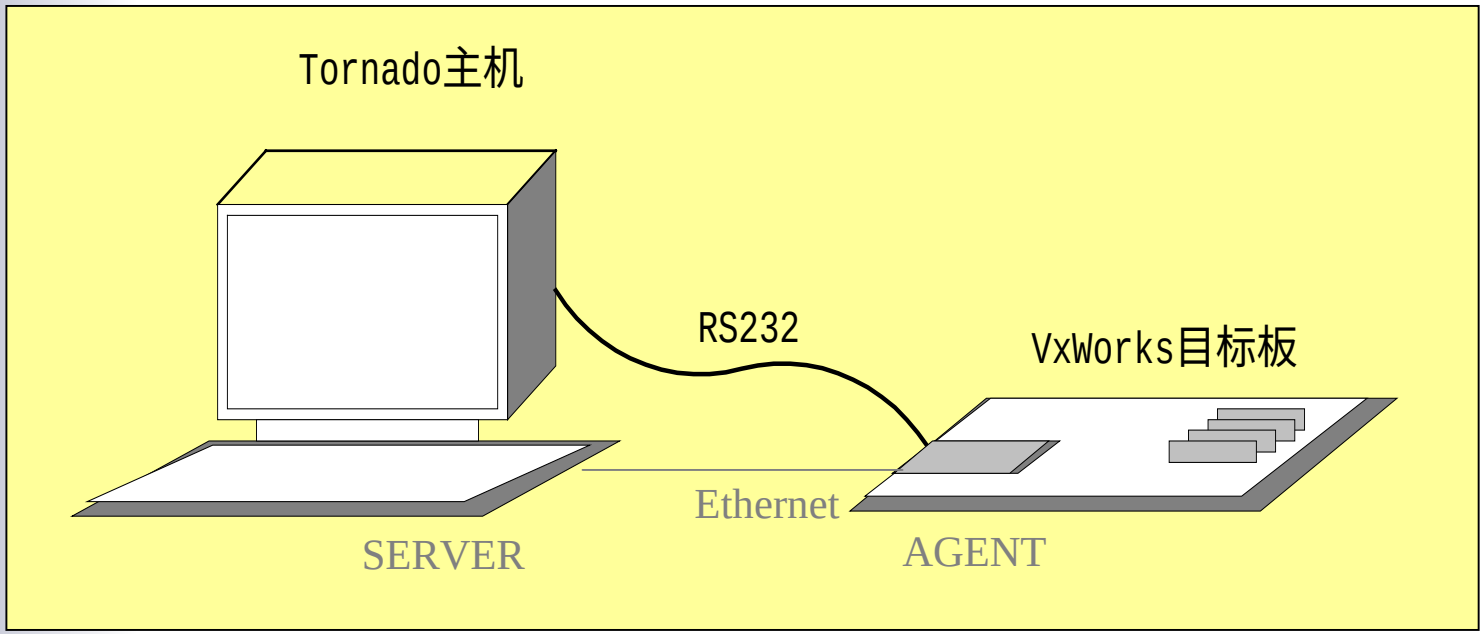
- 标准文本控制能力
- C和C++ 语法元素用不同颜色
- 调制集成：编辑窗口跟踪代码的执行
- 编译集成：项目管理将编译警告和编译错误直接和编辑窗口中的相应代码对应起来。

Tornado （2）

- Tornado集成开发环境由以下几个部分组成：

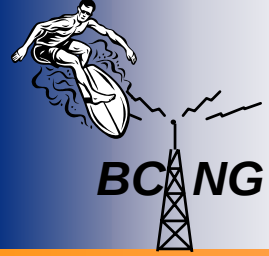


Vxworks/Tornado开发方式



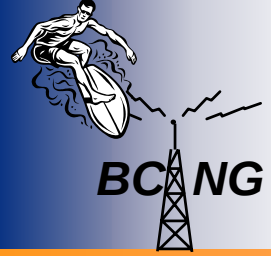
目标CPU： Vxworks 在其上运行 的一个单板计算机；

Tornado PC主机：有一根串行线与目标机相连（初始化时主机作为终端用），Vxworks核二进制文件驻留在其盘上，核的下载及运行Tornado 工具通过以太网口进行。



C++ 开发支持

- Tornado自带GNU C++编译器
- Tornado包括最新版本的iostream库和标准模板库的SGI实现
- Tornado的交互开发工具如Debugger、Shell等都支持C++
- Tornado还提供Wind Foundation Classes：
  - VxWorks Wrapper Class library
  - Tools.h++ library from Rogue Wave



目标机驻留工具

- 除了驻留在主机中的工具外，部分工具驻留在目标机中：
  - Target-resident shell
  - Symbal table
  - Module loader/unloader

工具库

- 中断处理支持：支持硬件中断和软件中断
- WatchDog定时器
- 消息日志：记录出错或状态消息
- 内存分配：可管理多个独立的内存池
- 字符串格式化和扫描：printf(), scanf()等
- 线形和环形缓冲区处理：可同时访问FIFO而不用互锁
- 链表操作：lstLib包含创建和处理双向链表的全集
- ASNI C库

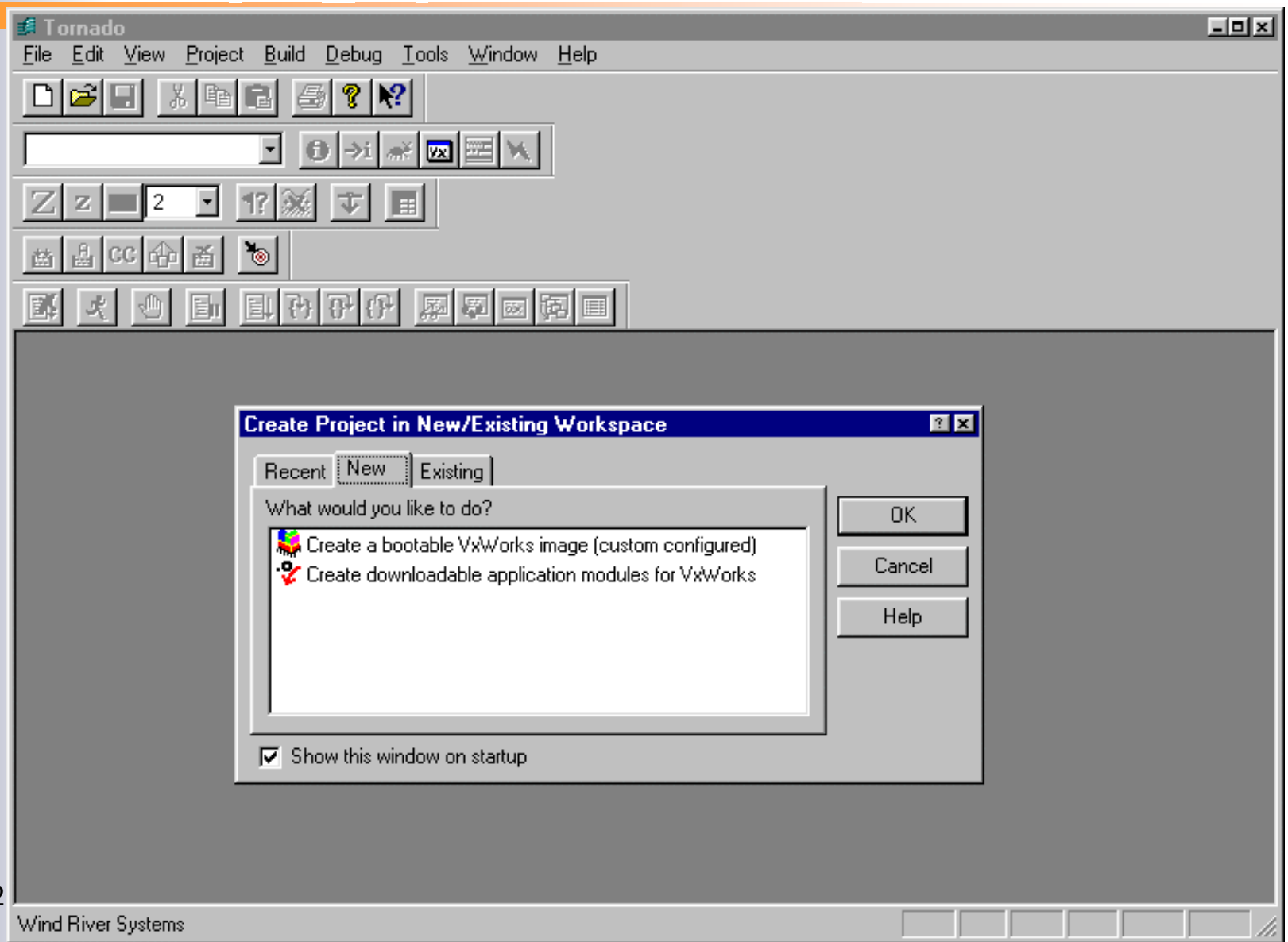
性能评估

- 执行计时器（Execution Timer）：测量程序运行时间
- Spy工具：提供每个task使用CPU的信息
  - 占用CPU的时间
  - 中断占用的时间
  - 空闲时间
- WindView提供更强大的监视功能

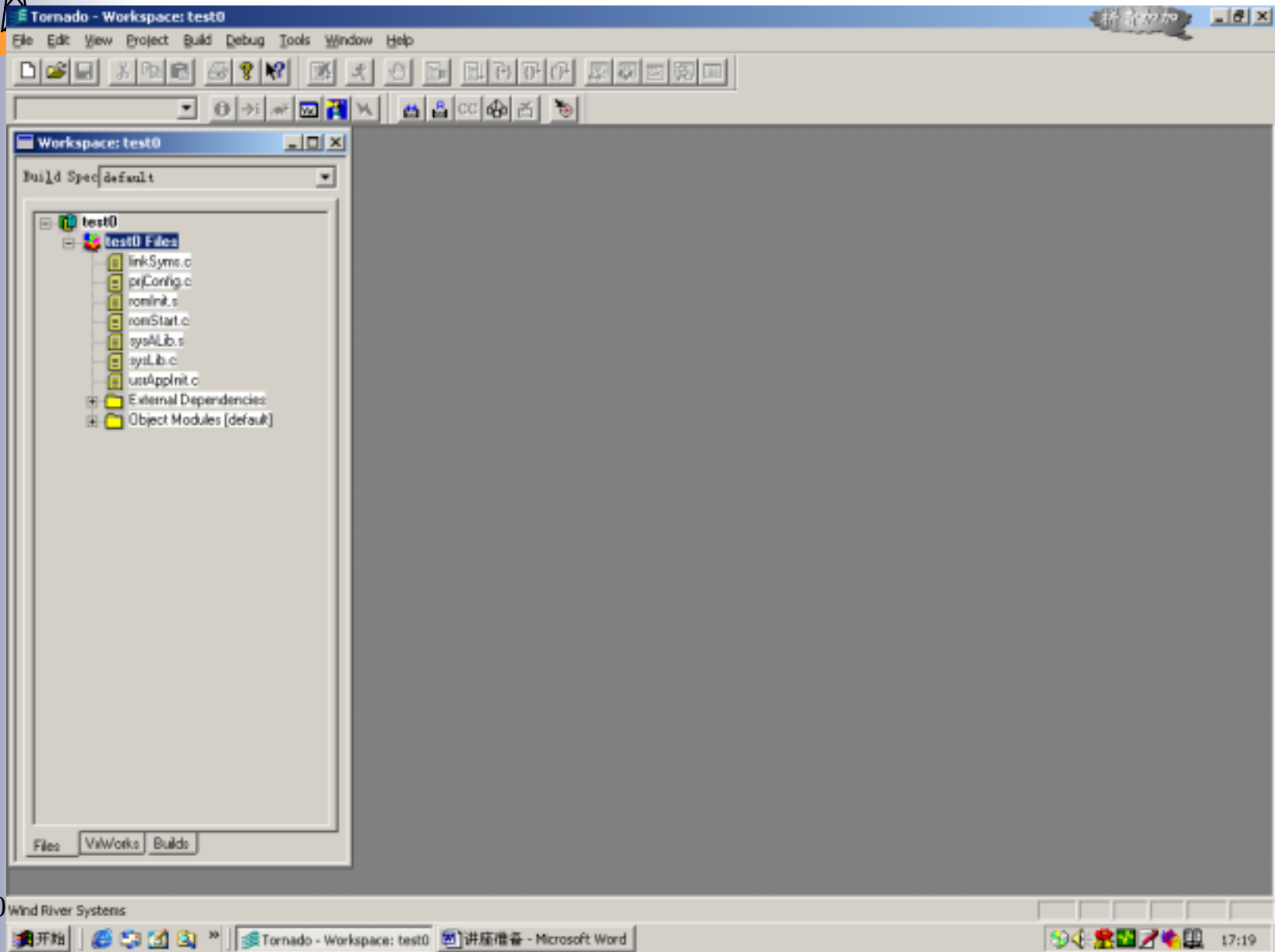
VxWorks仿真器（VxSim）

- VxWorks仿真器是一个模仿VxWorks目标机的程序，作为原型和测试环境
- 可以在一个主机上运行多个仿真器
- 不涉及到硬件驱动

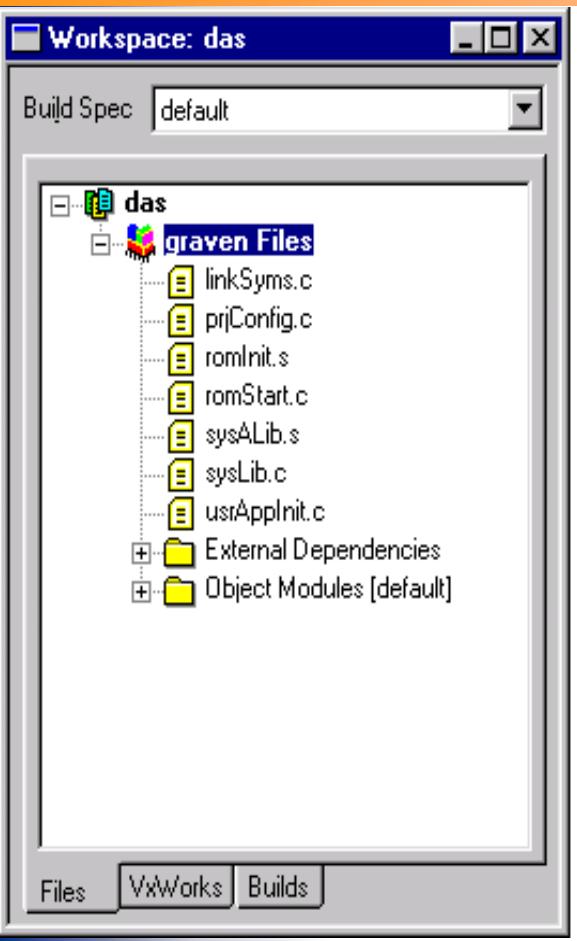
Tornado开发环境

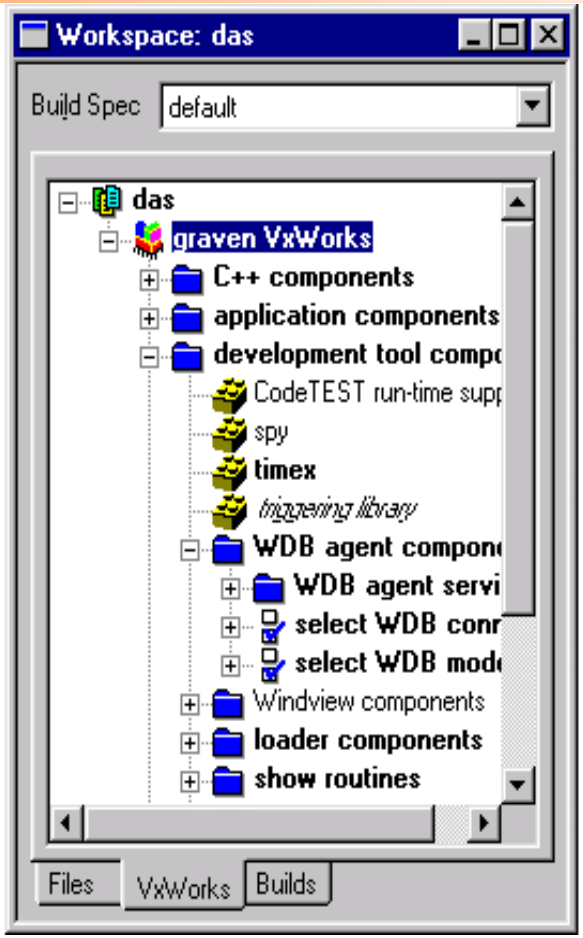


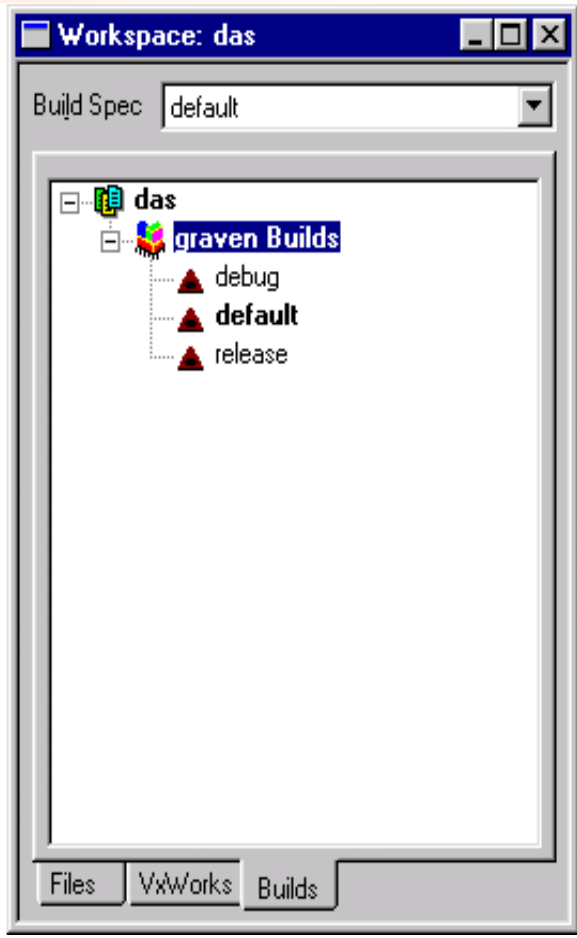
Tonado开发环境（续）



Tornado开发环境（续）



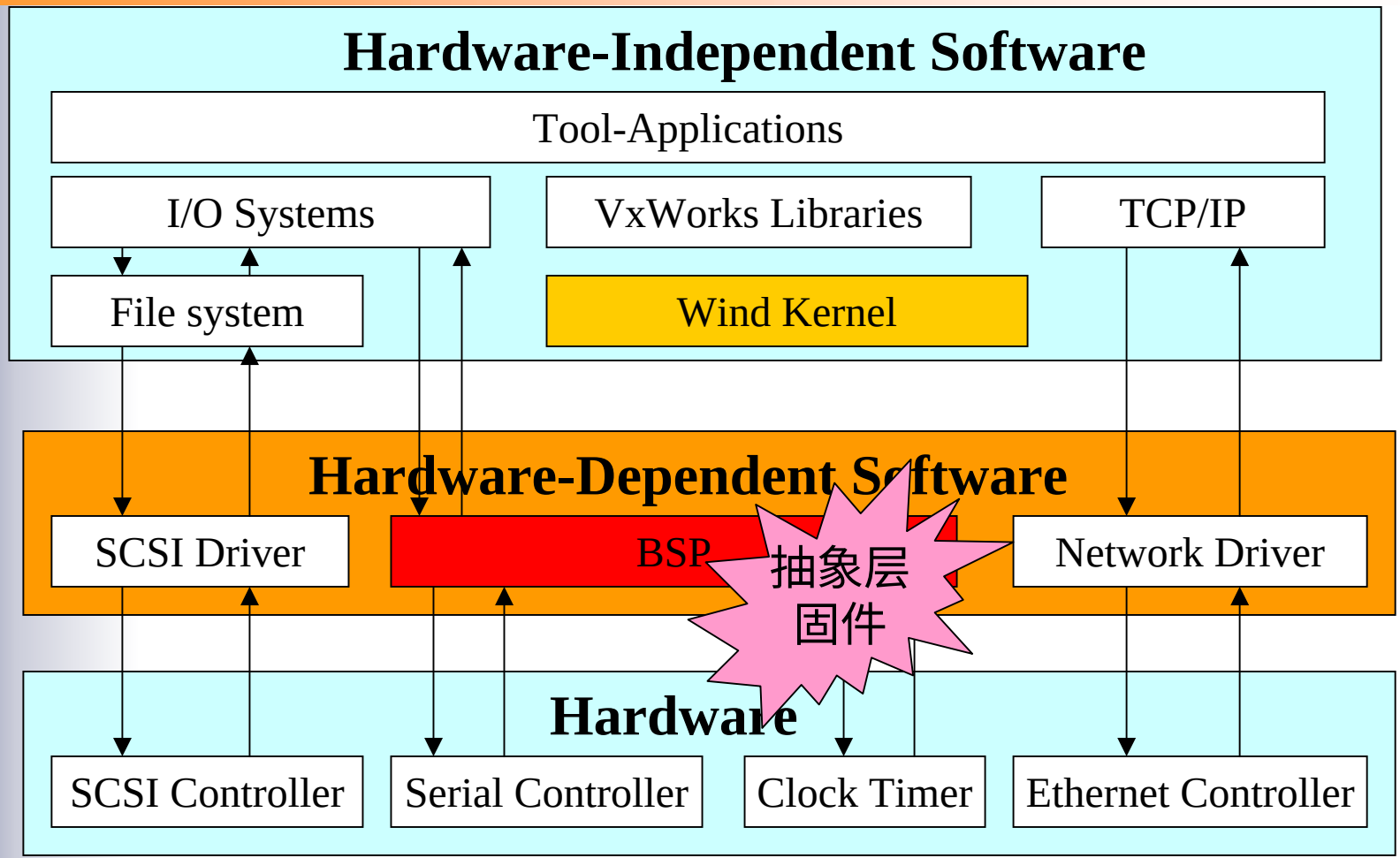




内　容

- 网络产品实现方法的演进
- 实时系统
- VxWorks介绍
- Tornado开发环境介绍
- BSP
- 设备驱动

VxWorks操作系统组成

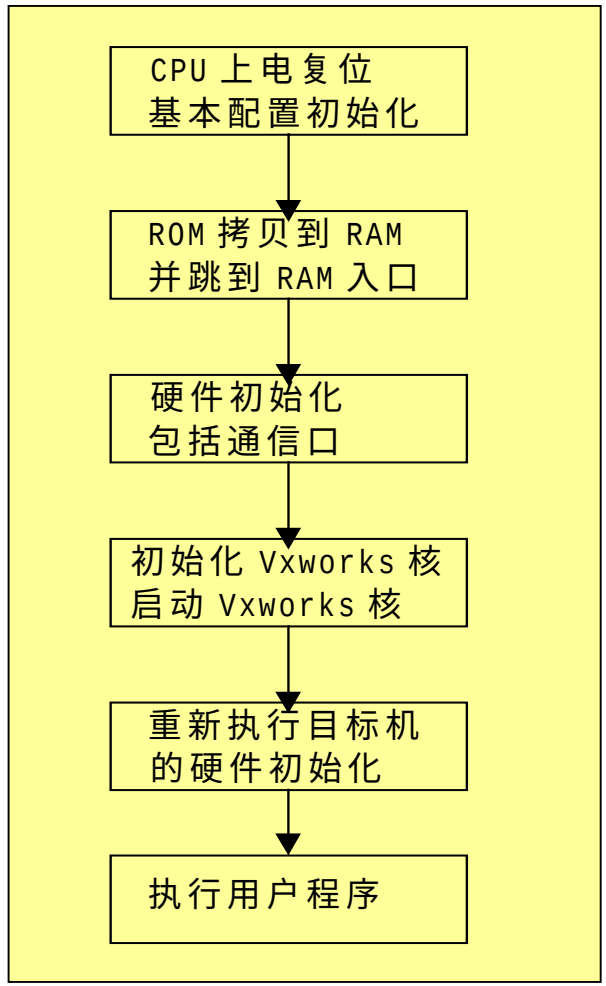


板支持包（BSP)

- 为各种目标板的硬件功能提供了统一的软件接口
- 它们包括：
  - 硬件初始化
  - 中断处理和产生
  - 硬件时钟和定时器管理
  - 内存映射和分配
- BSP还包括boot Rom和其它启动机制
- sysLib和sysALib库是VxWorks可移植的核心

板支持包（BSP）

Vxworks操作系统将一切与硬件有关的功能模块都放在BSP库中。该BSP库是硬件与软件的接口，处理硬件的初始化、中断处理与产生、硬件时钟与定时管理、局部和总线内存空间的映射、内存大小定义，等等。能够自行启动目标机、初始化目标机、能够与host通信以下载Vxworks核、把控制权交给Vxworks核来调用用户应用程序等功能。

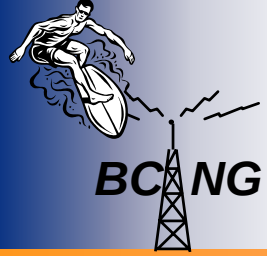


内　容

- 网络产品实现方法的演进
- 实时系统
- VxWorks介绍
- Tornado开发环境介绍
- BSP
- 设备驱动

VxWorks中的设备

- ttyDrv Terminal driver
- ptyDrv Pseudo-terminal driver
- pipeDrv Pipe driver
- memDrv Pseudo memory device driver
- nfsDrv NFS client driver
- netDrv Network driver for remote file access
- ramDrv RAM driver for creating a RAM disk
- scsiLib SCSI interface library
- - Other hardware-specific drivers



VxWorks与主机系统的I/O差别

- 设备配置：在VxWorks中设备驱动可以动态安装和卸载
- 文件描述字：在Windows和Unix中，fd是进程中唯一的；在VxWorks中fd是全局唯一的，标准输入输出例外（0，1，2）
- I/O控制：Unix和VxWorks中传给ioctl( )的参数可能不同
- 驱动例程：Unix下设备驱动运行于系统模式下，不可抢占；VxWorks下的设备驱动运行于线程模式下，可抢占。

内部结构

- 多数系统的设备驱动只提供少数低级I/O例程，如输入、输出等；大部分工作由I/O系统完成。
  - 驱动易实现
  - 设备动作尽可能类似
  - 驱动编写人员很难完成I/O系统未提供的协议
- VxWorks中I/O系统用于将用户请求交给适当的设备驱动，每个驱动根据自己的情况处理用户的I/O请求。
- VxWorks提供高级例程库用于设备驱动的编写
  - 为标准设备写驱动很容易，编码量小
  - 可以根据情况用非标准方式实现用户请求

设备驱动程序

- 驱动初始化例程xxDrv( )
  - 安装驱动、分配数据结构、连接中断服务例程、初始化硬件
- 设备创建xxDevCreate( )
  - 给驱动增加一个设备，参数包括缓冲区大小、设备地址等
  - 为设备初始化数据结构、信号量和硬件等
- 基本I/O功能
  - xxOpen()
  - xxRead()
  - xxWrite()
  - xxIoctl()
- 中断服务例程xxInterrupt( )

驱动安装

- I/O系统维护一个驱动表，将用户的I/O请求转换为适当的驱动程序
- 使用iosDrvInstall()动态安装驱动，参数为新驱动的7个I/O例程的地址，iosDrvInstall()将地址放入驱动表的空闲条目，返回此条目的索引，称为驱动号。
- 文件系统在驱动表中有自己的条目，它们在文件系统库初始化时创建。

驱动安装举例

DRIVER CALL:

```
drvnum = iosDrvInstall (xxCreat, 0, xxOpen, 0, xxRead, xxWrite, xxIoctl);
```

[1] Driver's install routine specifies driver routines for seven I/O functions.

[2] I/O system locates next available slot in driver table.

[3] I/O system enters driver routines in driver table.

[4] I/O system returns driver number (**drvnum** = 2).

DRIVER TABLE:

	create	remove	open	close	read	write	ioctl
0							
1							
2	xxCreat	0	xxOpen	0	xxRead	xxWrite	xxIoctl
3							
4							

设 备

- 一些驱动可以为某种设备的多个实例服务
- 在VxWorks中，数据结构设备头（DEV_HDR）定义设备
  - 设备名
  - 设备的驱动的编号（Index）
- DEV_HDR保存在驻留内存的设备列表（device_list）中
- DEV_HDR是设备描述字（device descriptor）的开始部分，后者包括特定设备的数据：
  - 设备地址
  - 缓冲区
  - 信号量
- 设备描述字只要以DEV_HDR开头便可，可包含任何与设备相关的信息

设备列表和增加设备

- 调用iosDevAdd()动态增加非块设备，参数为新设备的描述字的地址、设备名和驱动编号
- 驱动程序只需要填写描述字中与设备相关信息，不需要填写设备头；iosDevAdd()在设备头中填入设备名称和驱动编号，然后加入设备列表
- 增加块设备需要调用与此块设备文件系统相关的设备初始化例程，这个例程将自动调用iosDevAdd()

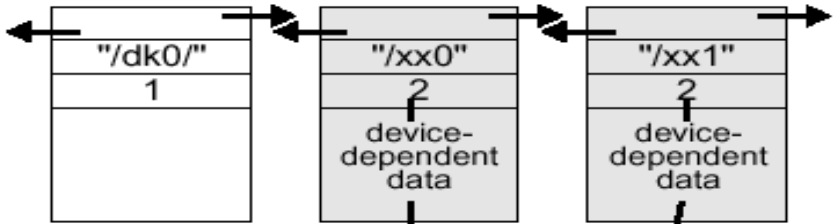
增加设备的例子

DRIVER CALLS:

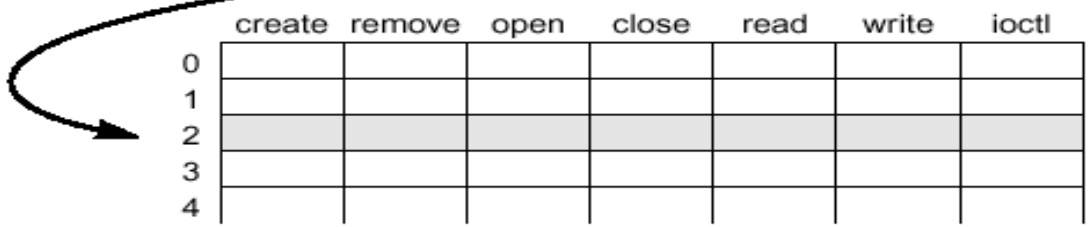
```
status = iosDevAdd (dev0, "/xx0", drvnum);
status = iosDevAdd (dev1, "/xx1", drvnum);
```

I/O system adds device descriptors to device list. Each descriptor contains device name and driver number (in this case 2) and any device-specific data.

DEVICE LIST:



DRIVER TABLE:



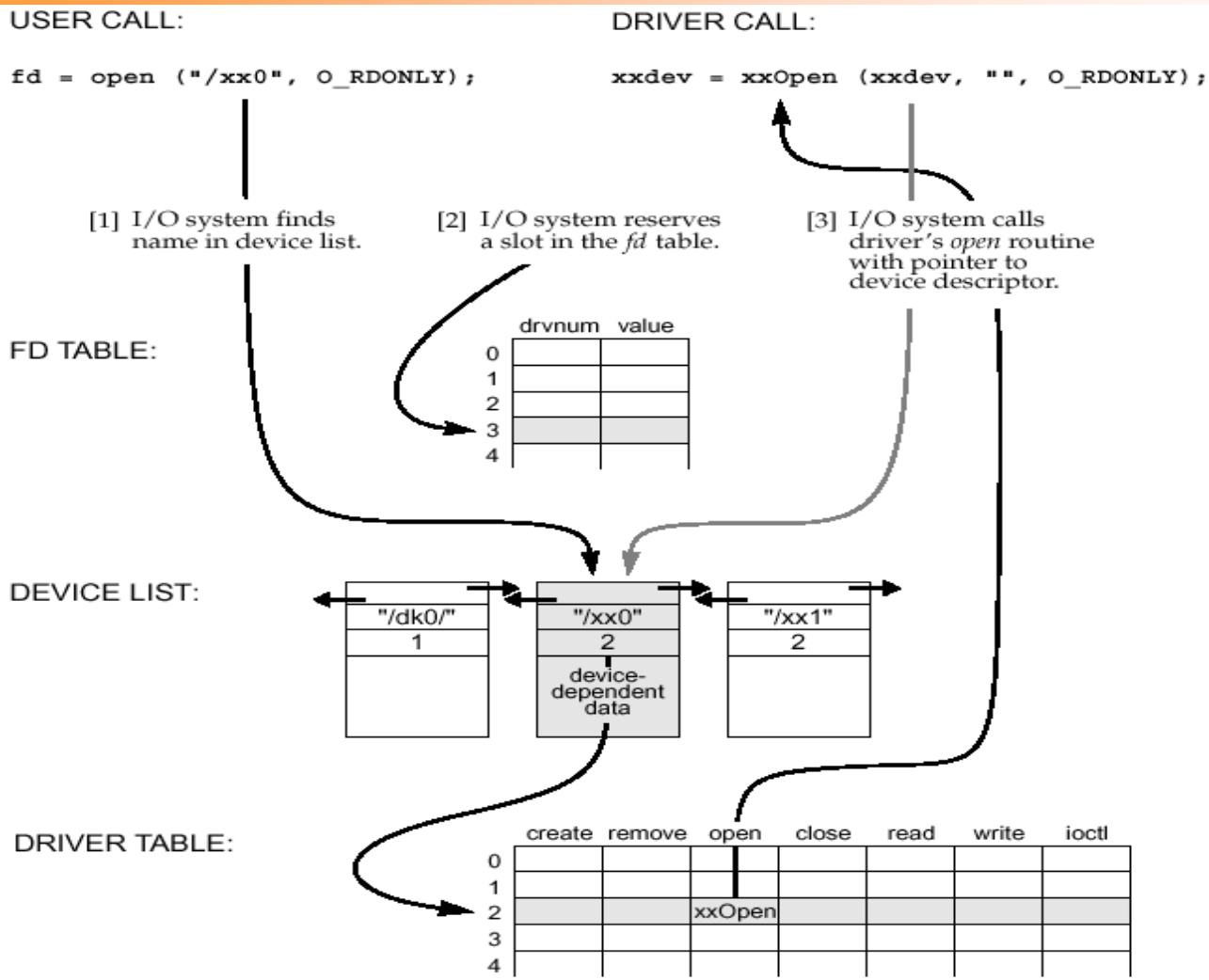
文件描述字

- 可以同时对一个设备打开多个fd
- 一个设备的驱动维护I/O系统设备信息和与fd相关的信息（如文件偏移量）
- 也可以对一个非块设备打开多个fd，如tty，这些fd没有附加信息，所以对它们的写操作效果相同

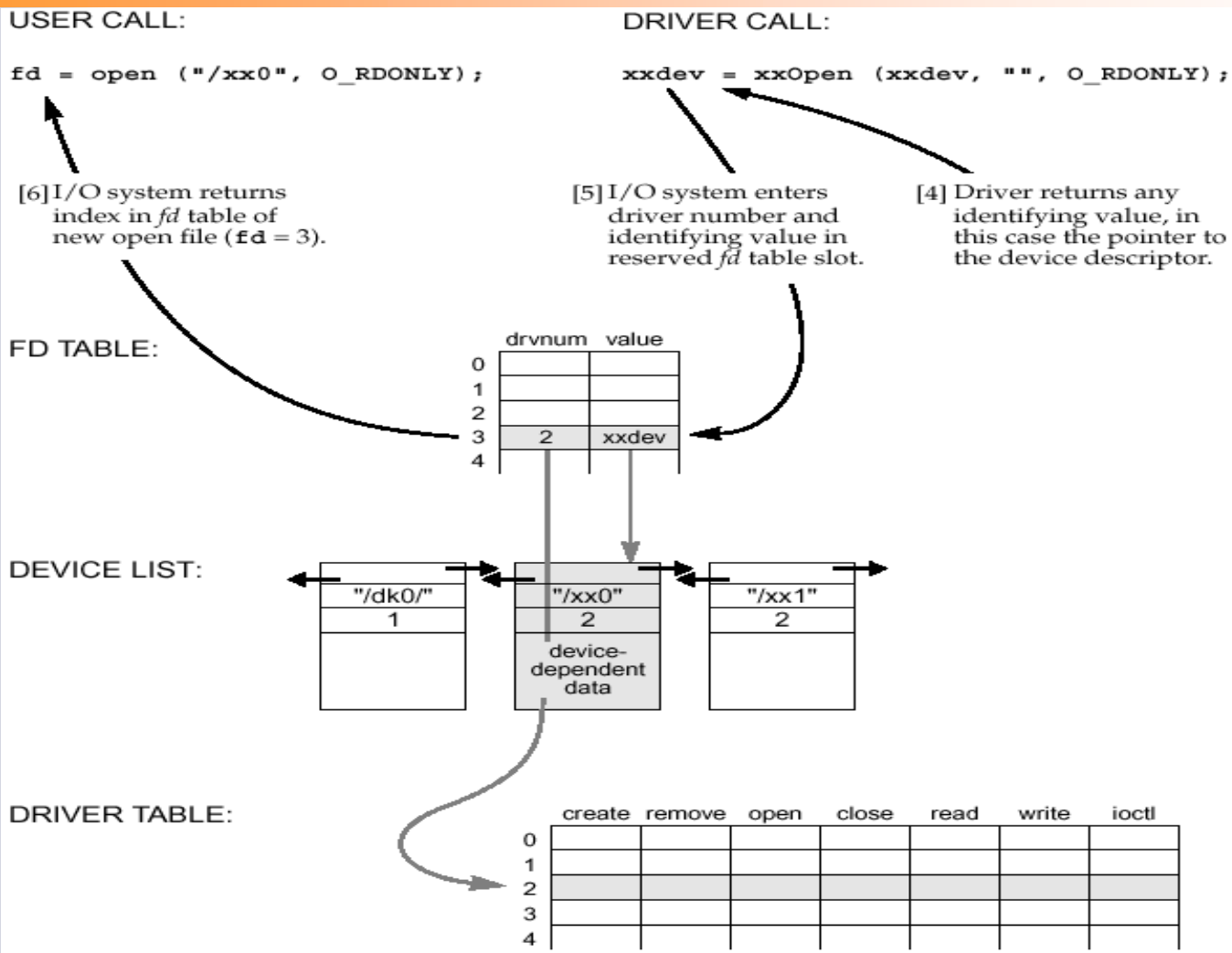
Fd表

- 文件用open()或creat()打开，I/O系统在设备列表中搜索与文件名最匹配的设备，找到后用设备头中的驱动编号查找对应的驱动表中的打开例程。
- I/O系统必须建立fd与驱动之间的联系
- 驱动必须将每个fd与特定数据结构关联起来，在非块设备的情况下，通常是设备描述字
- I/O系统在fd表中维护这些联系。表中包括
  - 驱动编号
  - 驱动确定的4字节值（用于标识文件）

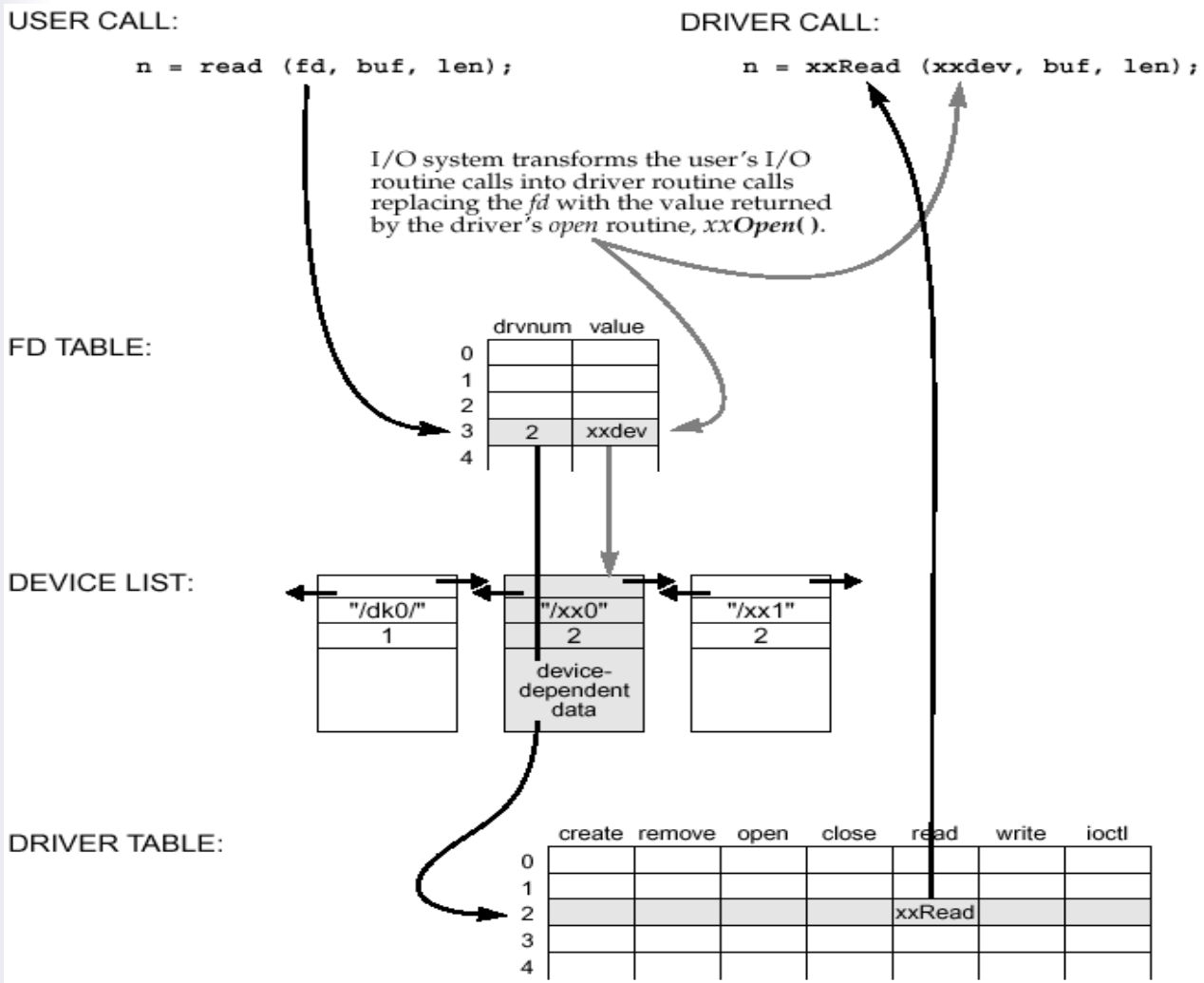
打开文件


USER CALL:
fd = open ("/xx0", O_RDONLY);
DRIVER CALL:
xxdev = xxOpen (xxdev, "", O_RDONLY);
[1] I/O system finds name in device list.
[2] I/O system reserves a slot in the fd table.
[3] I/O system calls driver's open routine with pointer to device descriptor.
FD TABLE:
drvnum
value
0
1
2
3
4
DEVICE LIST:
"/dk0/"
1
"/xx0"
2
device-dependent data
"/xx1"
2
DRIVER TABLE:
create
remove
open
close
read
write
ioctl
xxOpen

打开文件


USER CALL:
fd = open ("/xx0", O_RDONLY);
DRIVER CALL:
xxdev = xxOpen (xxdev, "", O_RDONLY);
[6] I/O system returns index in fd table of new open file (fd = 3).
[5] I/O system enters driver number and identifying value in reserved fd table slot.
[4] Driver returns any identifying value, in this case the pointer to the device descriptor.
FD TABLE:
drvnum
value
0
1
2
3
2
xxdev
4
DEVICE LIST:
"/dk0/"
1
"/xx0"
2
device-dependent data
"/xx1"
2
DRIVER TABLE:
create
remove
open
close
read
write
ioctl
0
1
2
3
4

从文件中读取数据


USER CALL:
n = read (fd, buf, len);
DRIVER CALL:
n = xxRead (xxdev, buf, len);
I/O system transforms the user's I/O routine calls into driver routine calls replacing the fd with the value returned by the driver's open routine, xxOpen( ).
FD TABLE:
drvnum
value
0
1
2
3
2
xxdev
4
DEVICE LIST:
"/dk0/"
1
"/xx0"
2
device-dependent data
"/xx1"
2
DRIVER TABLE:
create
remove
open
close
read
write
ioctl
0
1
2
xxRead
3
4

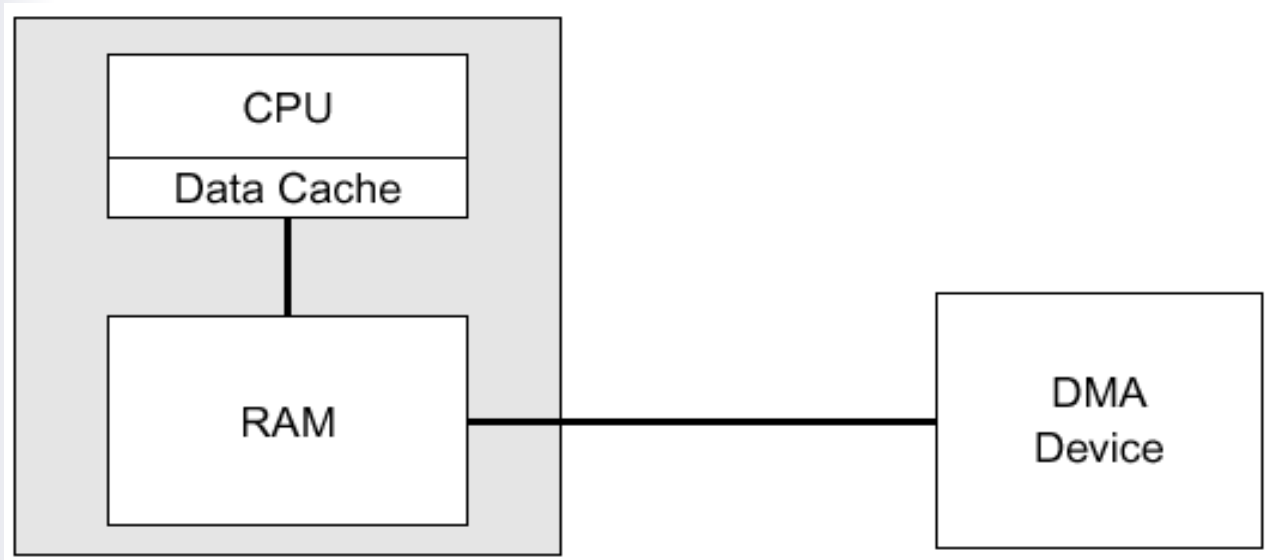
关闭文件

- 用户使用close()关闭文件
- I/O系统根据fd表找到对应驱动的关闭例程
- 驱动的关闭例程运行之后，I/O系统将fd表中对应条目标识为可用

实现select()

- Select()可以使一个task等待在多个I/O上，或者超时返回。
- 设备驱动支持select()的方法和步骤，参见VxWorks编程指南（VxWorks Program Guide）的3.9.3节

Cache一致性



Cache一致性

- 数据Cache通过减少内存访问次数来提高性能
- 有cache的板子的驱动必须保证Cache的一致性
- Cache一致性指Cache与RAM中的数据必须同步或一致
- 当出现对RAM的异步访问时（如DMA设备访问或VME总线访问），Cache和RAM中的数据可能会失去同步
- 数据cache有两种工作方式：
  - Writethrough：向cache和RAM写数据，保证输出同步，不保证输入
  - Copyback：只向cache写数据，不能保证输出或输入同步

Cache一致性

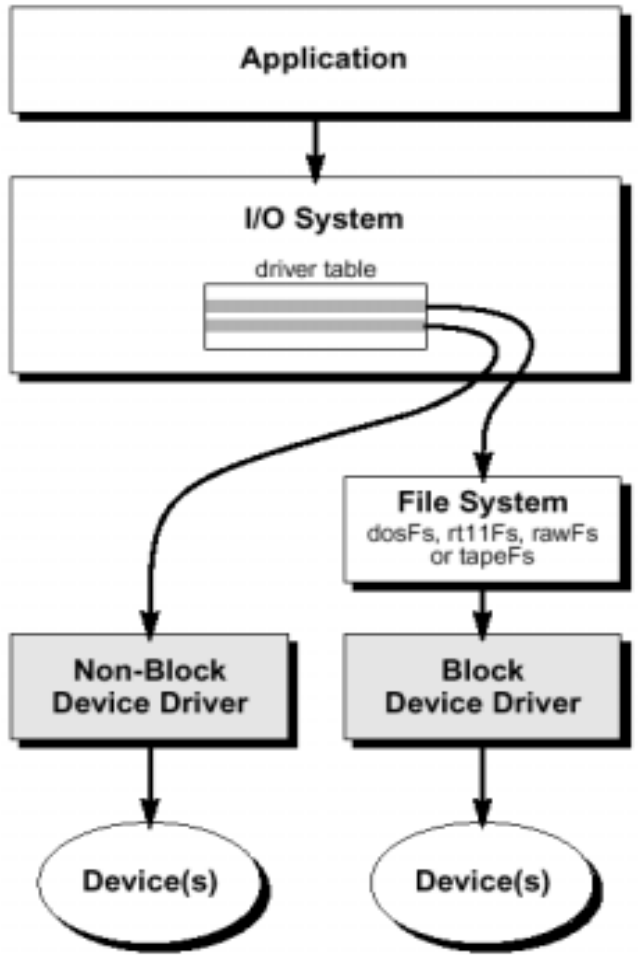
- 在Copyback的情况下，如果DMA从RAM中读数据，可能会与Cache中不一致。所以读之前要保证Cache中的数据全部刷新到RAM中
- 如果CPU要读取来自DMA设备中的数据，那么从RAM和Cache中读的数据可能不一致，因此要将Cache中的数据标识为非法，使CPU从RAM中读取数据
- 驱动保证Cache一致性的方法：
  - 分配cache安全缓冲区（不能cache的缓冲区）
  - 当向设备写数据或从设备读数据时，刷新cache或标识为非法

Cache一致性

- 分配cache安全缓冲区
  - 对静态缓冲区有用，但要求MMU支持
  - 经常分配或释放不能cache的缓冲区（动态缓冲区）将导致大量内存被标识为不能cache
- 手动刷新Cache条目或者将其标识为非法，可以使动态缓冲区保持一致
  - CacheFlush()
  - cacheInvalidate()
- 将上面两种方法结合起来效率更高
  - 只有非常必要时才刷新Cache条目或将其标识为非法

块设备

- 在VxWorks中，块设备不是直接与I/O系统交互，而是通过文件系统与I/O系统交换
- 从SCSI-1开始支持直接访问块设备，与各种操作系统兼容
- VxWorks还支持SCSI-2顺序设备，数据块只能写在媒质末尾，不能替换中间的数据，但可以从任何地方读取数据；这与其它块设备的处理不同

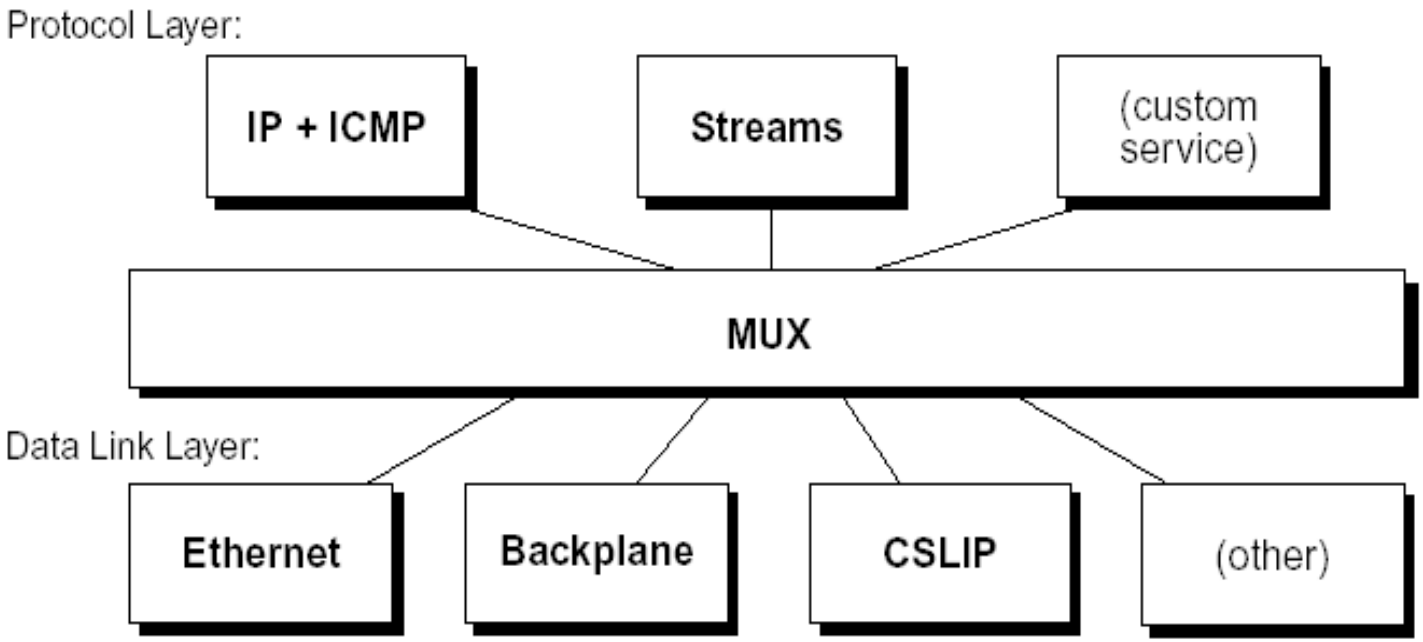


块设备驱动

- 块设备驱动必须支持创建逻辑块设备结构，包括一些公共特性，如设备物理配置变量、指定驱动的例程等
  - BLK_DEV：直接访问块设备
  - SEQ_DEV：顺序块设备
- 设备初始化（dosFsDevInit()）
- 块设备的低级驱动不在I/O系统的驱动表中，而是每个文件系统作为一个“驱动”安装在驱动表中
  - 每个文件系统在驱动表中只占一个条目，即便它为多个设备服务
- 设备初始化后，与某文件系统关联，所有I/O操作经文件系统查找BLK_DEV或SEQ_DEV中的例程

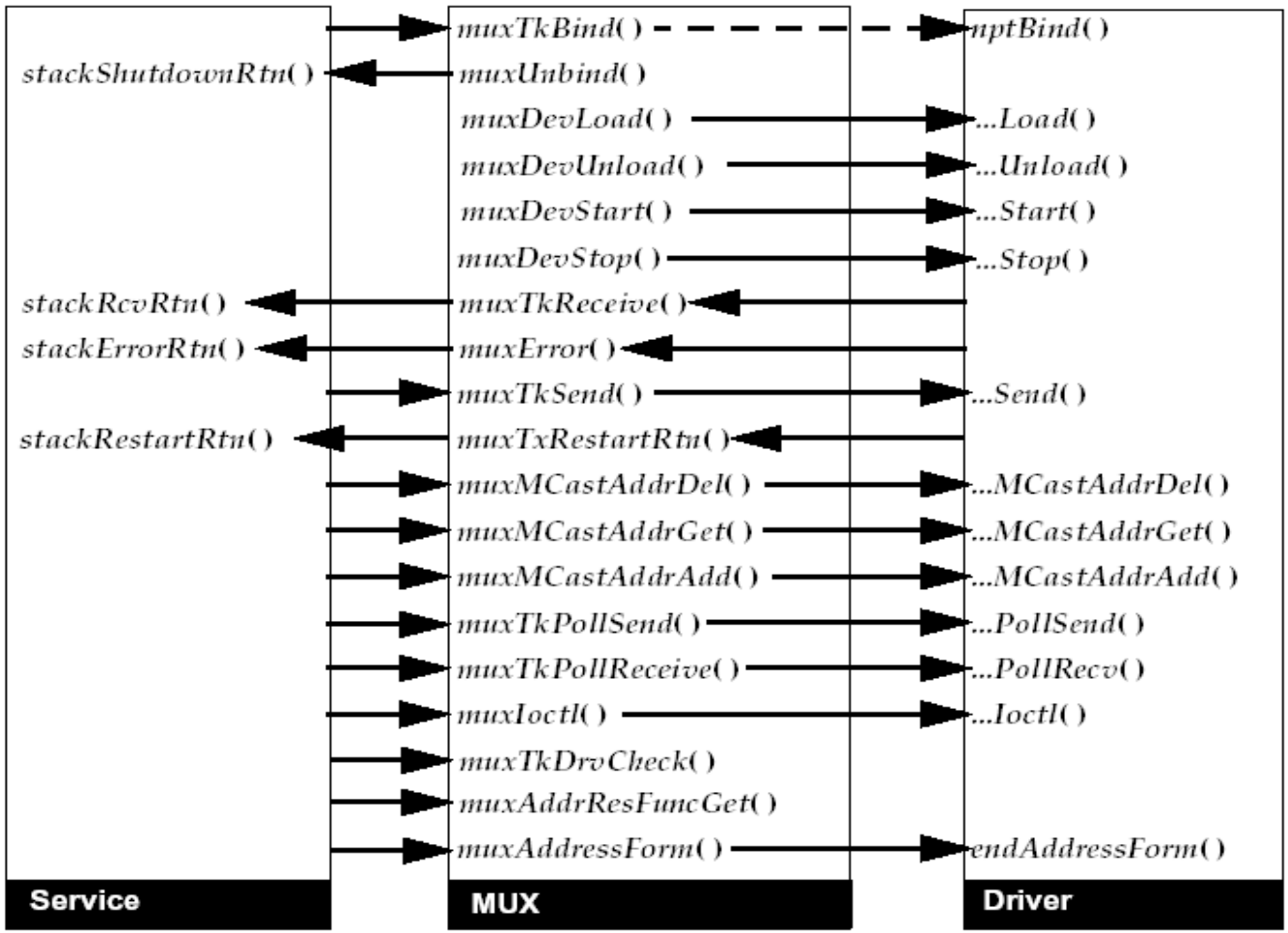
NPT套件

Figure 1-2 **The MUX Is the Interface Between the Data Link and Protocol Layers**

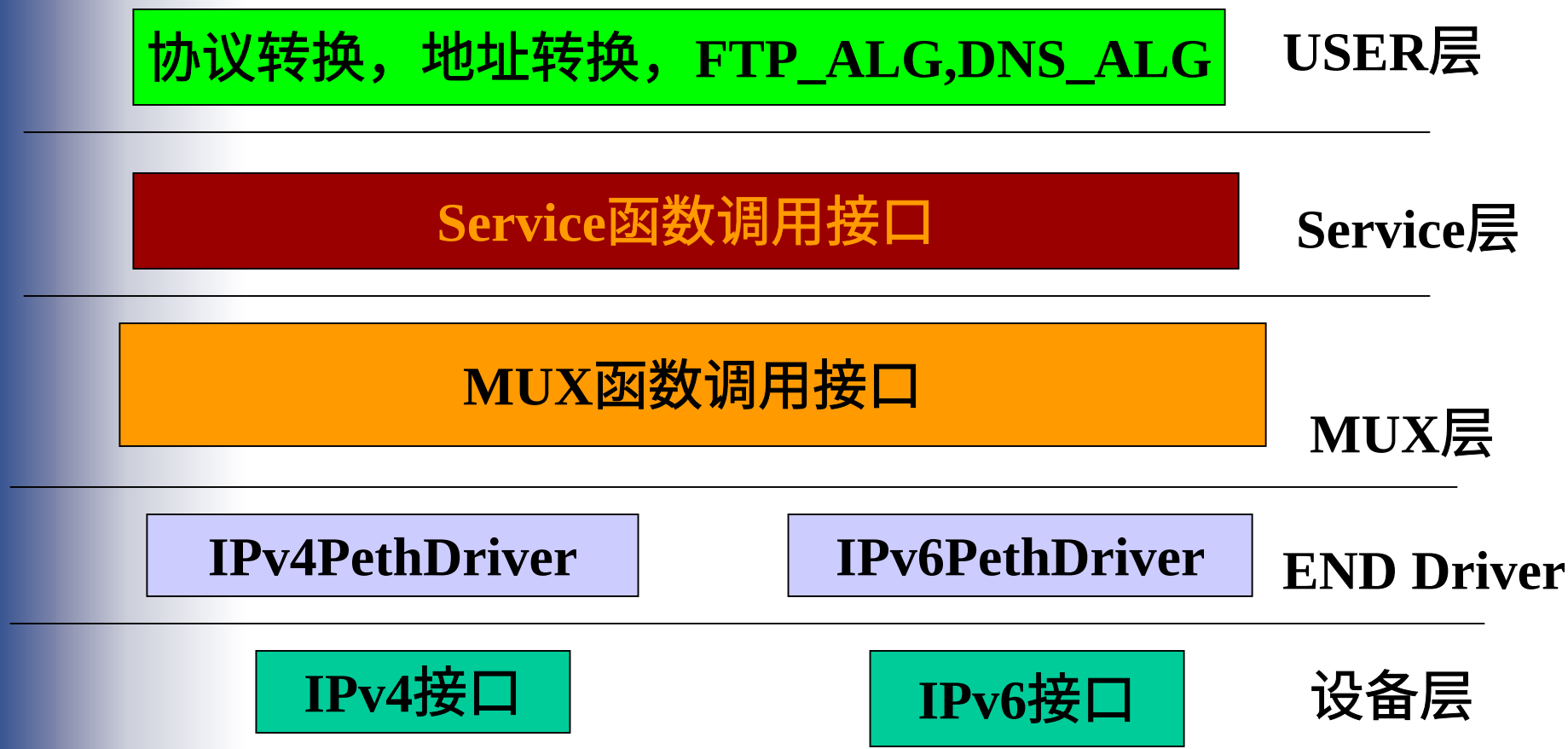


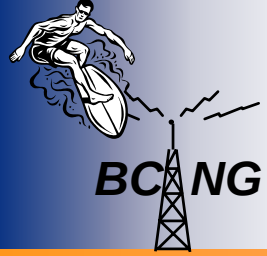
NPT套件

Figure 1-3 **The MUX Interface**



使用NPT收发包的层次结构





谢 谢！

- 欢迎加入宽带通信网络研究组！