第三章

嵌入式软件系统

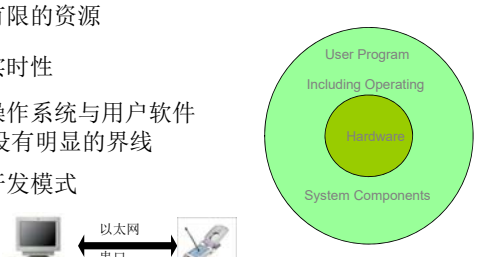
主要内容

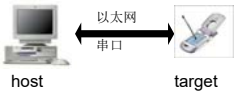
- 嵌入式软件系统概况
- 嵌入式操作系统
- 常见的嵌入式操作系统
- 嵌入式软件开发和运行中的几个技术

嵌入式软件系统的分类

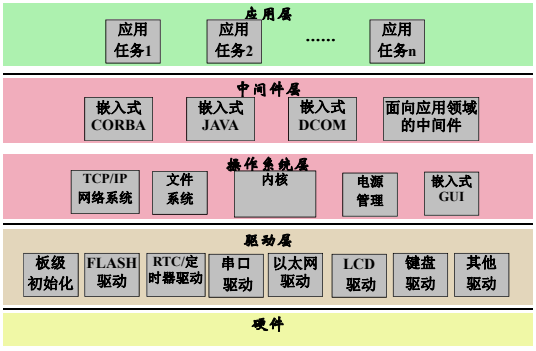
嵌入式系统软件的特点

- 有限的资源
- 实时性
- 操作系统与用户软件没有明显的界线
- 开发模式

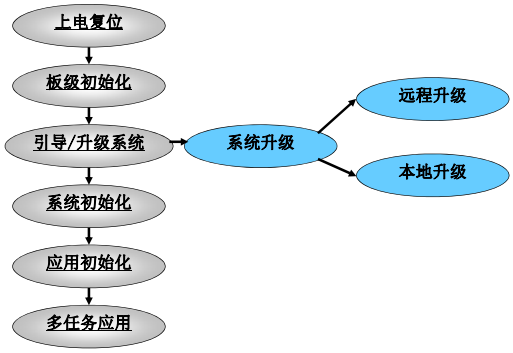




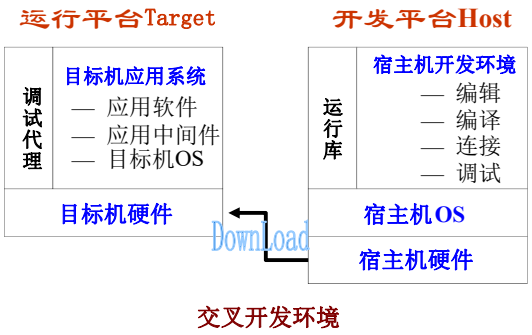
嵌入式软件体系结构



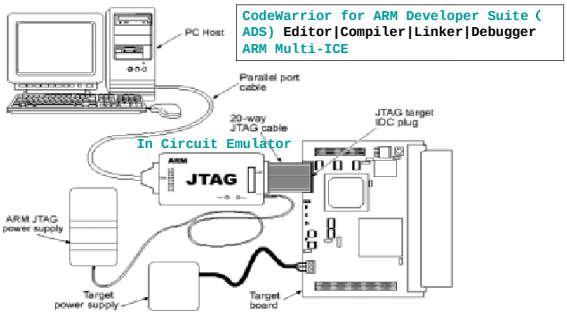
嵌入式软件运行流程



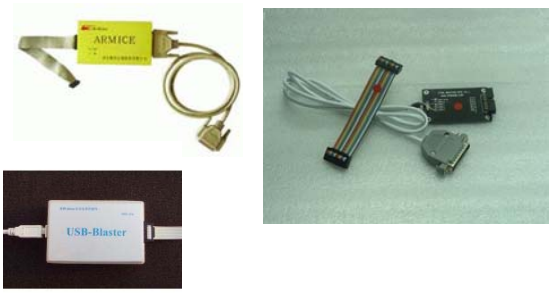
嵌入式软件的交叉开发环境

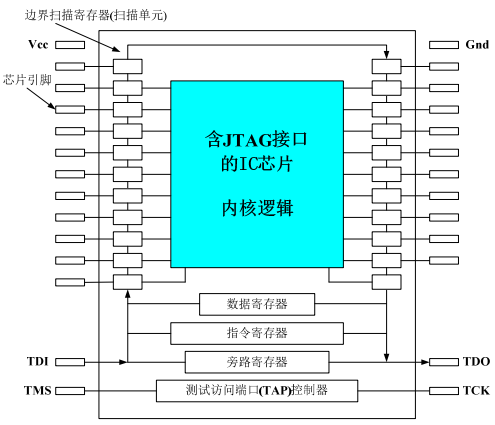


应用系统开发环境

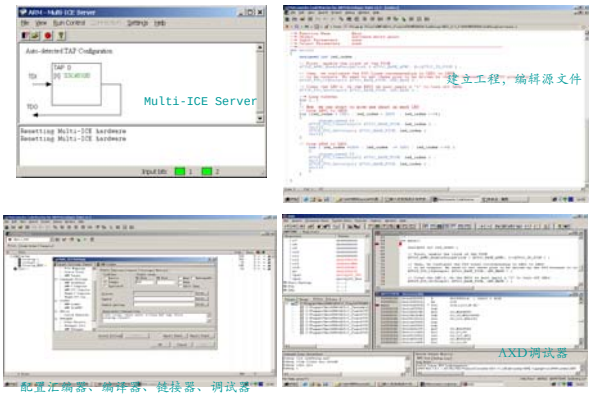


JTAG仿真器





应用系统开发软件



主要内容

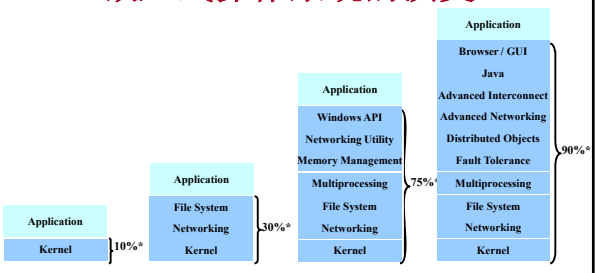
- 嵌入式软件系统概况
- **嵌入式操作系统**
- 常见的嵌入式操作系统
- 嵌入式软件开发和运行中的几个技术

嵌入式操作系统EOS
Embedded Operation System

一定需要嵌入式操作系统吗？

- “单片机+控制”应用 往往没有操作系统
- 简单的系统不一定需要操作系统
  - 单一任务，没有复杂的设备需要控制
  - 没有复杂的网络/通信协议需要实现
  - 没有太多的数据存储、文件系统访问需求
- 操作系统需要更多的资源
  - 复杂应用起码几百K级别的RAM，几MB级别的ROM，几M级别的CPU clock

嵌入式操作系统的演变



*Percent of total software supplied by RTOS vendor in a typical embedded device

嵌入式操作系统特征

- 小巧
  - 受硬件限制，必须小巧（最小几K）
- 实时性
  - 实时性是EOS的重要指标（us级响应）
- 可裁剪
  - 根据应用的需要进行裁剪，去掉多余部分

嵌入式操作系统特征

- 代码固化
  - EOS和应用软件被固化在ROM中
- 文件管理
  - 内存文件系统
- 弱交互性
  - 很多应用不需要（or 无人值守）人工干预
- 强稳定性
  - 运行不需人工干预，要求较高的稳定性

实时操作系统的特性

- 实时系统：采用各种算法和策略，始终保持系统行为的可预测性
  - 不仅逻辑上正确，还必须在时间上及时
  - **注重个体表现：与通用OS不同，要求每个实时任务在最坏情况下都要满足实时要求**
- 非实时系统
  - **逻辑正确即可**
  - **关心系统的平均响应时间（吞吐量）**

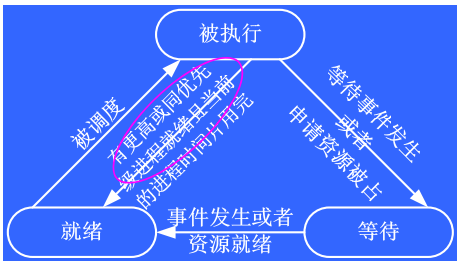
- 软（Soft）实时和硬（Hard）实时
  - 硬实时 – 过时的响应 →完全无用甚至产生致命后果。
    - 如ABS刹车
  - 软实时 – 过时的响应 → 服务质量下降。
    - 如视频通信系统
- 实时系统和高性能系统
  - 实时系统不一定是高性能，反之亦然
  - 高性能系统追求系统的总体效率，比如一秒能处理多少个事务（Transaction）。
  - 实时系统追求系统的可预测性（处理一个事务最差情况下需要多少时间）

嵌入式实时OS的主要性能指标

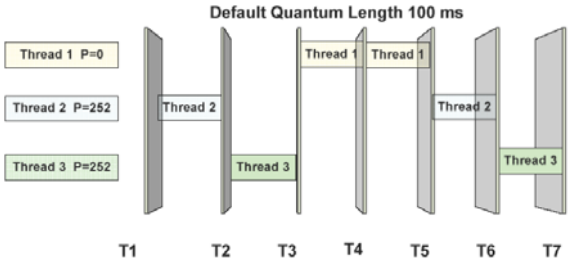
表 2 系统实时性能重要指标的典型值	VxWorks	uC/OS II	RT-Linux2.0	QNX6
硬件平台	MC68000	33MHz 486	60MHz 486	33MHz 486
任务切换	3.8us	<9us	不详	12.57us
中断响应	<3us	<7.5us	25us	7.54us

抢占式(Preemptive)多任务调度策略

- 每个任务分配优先级
  - 高优先级任务强制将低优先级任务切换出CPU
  - 高优先级任务一直执行完毕，再执行低优先级任务



多任务和调度



抢占式RTOS优先级分配方法

- 问题 – 怎样为一组跑在同一CPU上且有实时性要求的任务分配各自的优先级，才能最大程度保证它们满足各自的截止时限？
- 简化的模型（无资源竞争）
  - 任务之间没有互斥访问共享资源的要求
  - 实时性要求：每隔T时间，任务必须启动并完成一次，且其deadline正好为每个周期的末尾
  - 任务每次启动后需要在CPU上执行C时间才能执行完成

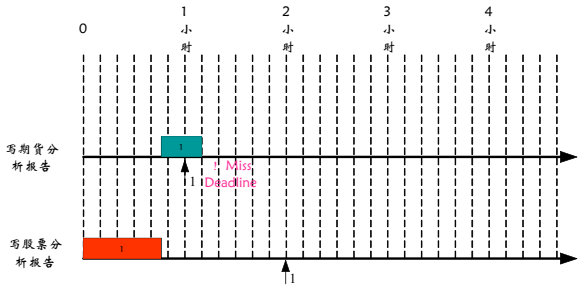
- 静态优先级分配
  - 优先级事先由开发者确定好
  - RMS(Rate Monotonic Scheduling，单一速率调度)方法
- 动态优先级分配
  - 优先级不是事先由开发者确定好，而是由OS Kernel来分配
  - EDF(Earliest Deadline First，截止时限最近者优先)方法
    - OS每次在调度时机到来时，计算每个任务离其各自截止时限的时间Td，Td最小的那个任务被调度执行

RMS(单一速率调度)优先级分配方法

- 基本准则
  - 重复周期越短(即重复率越高)的任务，分配的优先级越高
- 实例
  - 要求每小时交一篇最新期货市场分析报告，每两小时交一篇最新股票市场分析报告
  - 写期货报告需要25分钟，写股票报告需要45分钟

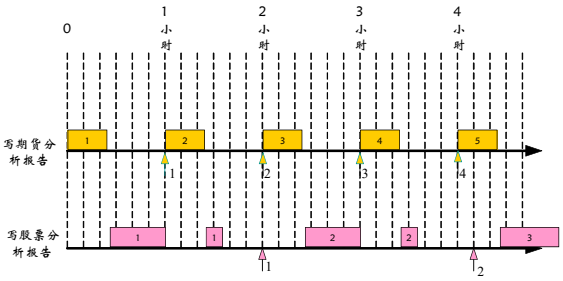
实例

- 优先写股票报告，将出现无法满足截止时限的问题



采用RMS思想分配优先级

- 优先写期货报告，可以满足截止时限要求



WinCE线程优先级分布图(例)

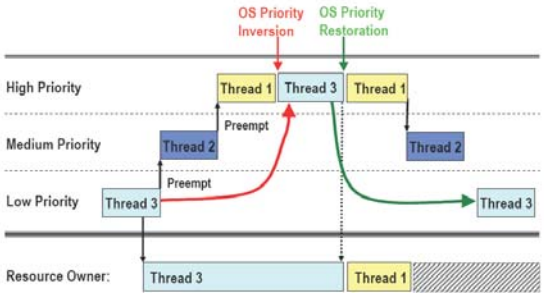
Priority	Component
0-19	Open – Real Time Above Drivers
20	Graphics Vertical Retrace
99	Power management Resume Thread
100-108	USB OHCI UHCI, Serial
109-129	IRSIR1, NDIS, Touch
130	KITL
131	VMini
132	CxPort
145	PS2 Keyboard
148	IRComm
150	TAPI
248	Power Management
249	WaveDev, Mouse, PnP, Power
250	WaveAPI
251	Normal
252-255	Open - Applications

优先级高于普通驱动的实时应用

基于WindowsCE的驱动程序

非实时的普通APP

任务存在资源竞争时
- 优先级反转和继承



RTOS应该具备的功能

- 多任务，不同任务具有不同优先级
- 任务是可抢占的
- 具有消除优先级反转的机制
- OS的中断延迟、任务切换、驱动程序延迟行为是可知和可预测的
- 支持多任务间的通信，通信延迟可预测
- ……

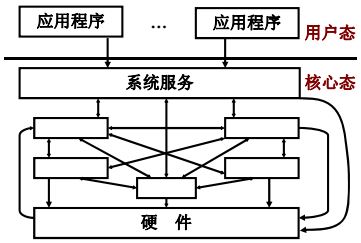
RTOS的性能对比

RTOS使用技术	VxWorks	UC/OS-II	RT-Linux	QNX6
供应商	Wind River	Micrium	FSMlabs	Quanturm
抢先式内核	是	是	是	是
调度算法	优先级时间片轮转	优先级	优先级最短时限优先	优先级先进先出循环
优先级分配	动态	静态	动态	动态
优先级继承	是	无	无	是
优先级数	256	64	不限	32
时间确定性	是	是	是	是

嵌入式操作系统体系结构

- 体系结构：定义了硬件与软件的界限、内核与操作系统其它组件（文件、网络、GUI等）的组织关系、系统与应用的接口。
- 目前操作系统的体系结构可分为：
  - 单块结构
  - 层次结构
  - 客户/服务器结构

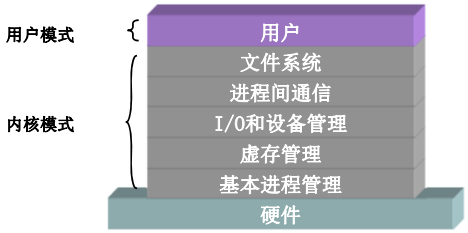
嵌入式操作系统体系结构-单块结构



- 优点 - 一次系统调用只有一次特权态切换，效率高
- 缺点 - 操作系统实现复杂，鲁棒性、安全性难保证（相对而言）

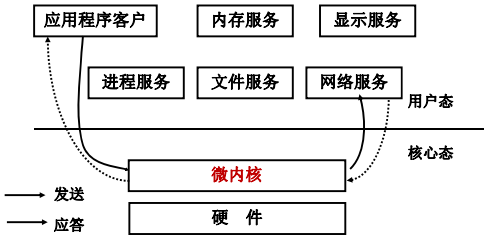
– Linux，类UNIX系统、MS Windows XP

嵌入式操作系统体系结构---层次结构



层次结构（胖内核）:Windows CE 6.0

嵌入式操作系统体系结构---- 微内核

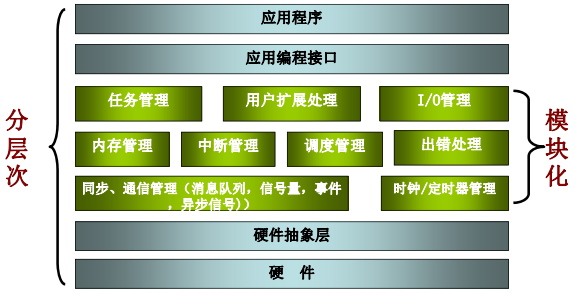


微内核（瘦内核）：进程调度、进程间通信、内存管理
好处 - 实现简单、易于保证鲁棒性和安全
坏处 - 一次系统调用需要两次进程上下文切换，效率低

嵌入式操作系统体系结构

- 目前嵌入式操作系统主要采用分层和模块化相结合的结构或微内核结构。
  - 分层和模块化结合的结构将操作系统分为硬件无关层、硬件抽象层和硬件相关层，每层再划分功能模块。
  - 移植工作集中在硬件相关层，与其余两层无关，功能的伸缩则集中在模块上，从而确保其具有良好的可移植性和可伸缩性。

嵌入式操作系统体系结构



层次＋模块结构

嵌入式RTOS的现状和发展方向

- 要求RTOS具有网络和图形界面功能
- 支持标准C/C++ API，开发与Windows、Unix保持一致
- 从微内核向超微内核发展
  - Microkernal：将传统OS的共性功能抽象出来，构成OS的功能基础
  - Nanokernal：实时超微内核，提供可抢占、快而准确的实时服务，可重用可伸缩
- 开发环境向开放集成的方向发展
  - 第三代：客户-服务器结构
  - 运行系统、连接的无关性、开放的软件接口

主要内容

- 嵌入式软件系统概况
- 嵌入式操作系统
- 常见的嵌入式操作系统
- 嵌入式软件开发和运行中的几个技术

常见的嵌入式操作系统

- 从开源角度划分（开源or闭源）
  - 闭源的是商业的，用户不仅不能修改源代码，而且使用时需要付费。
  - 有些系统的源代码虽然公开，但并不意味着免费，如μC/OS-II。
  - Linux系列的操作系统由于开源，在嵌入式领域应用面很广。

嵌入式操作系统分类

- 按收费模式划分
  - 商用型
    - Vxworks, Nucleux， PlamOS, Symbian, WinCE, QNX, pSOS,VRTX,Lynx OS, Hopen, Delta OS
  - 免费型
    - Linux, μCLinux, μC/OS-II,eCos, Uitron
- 按实时性划分
  - 硬实时（面向控制和通信等领域）
    - VxWorks
  - 软实时（面向消费电子产品）
    - WinCE, RTLinux

常见的嵌入式操作系统

- VxWorks
- QNX
- Nucleus PLUS
- uC/OS II / III
- Palm OS
- Windows CE
- 各类嵌入式Linux
- ……

VxWorks

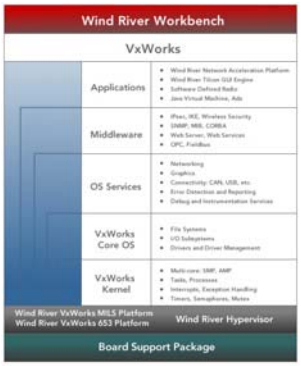
- 在实时性嵌入式操作系统中市场份额最高，由WindRiver公司（现为Intel子公司）开发。
  - 1985年开始
- 目前最新版本为7.0
- 高可靠，广泛用于通信、军事、航空、航天等领域
  - F-16战斗机、B-2隐形轰炸机、爱国者导弹、几乎所有火星探测器
  - 原先对中国禁止销售，自解禁以来，在我们的军事、通信、工业控制等领域得到了非常广泛的应用。
- 支持x86, PowerPC, MIPS, ARM, SH-4等CPU结构
- 使用者需要支付
  - 一次性许可费（ License Fee）
  - 每个卖出系统的专利使用费（ Royalty Fee）

VxWorks特点

所需存储空间或称代码空间

- 高可靠性
- 高效硬实时
  - 微秒级的中断响应
  - 256个优先级的抢占式多任务调度
    - 同级别任务可以选择轮转式调度
  - 基于优先级继承方式解决优先级反转问题
  - 快速、确定的进程上下文切换时间
- 系统footprint小，可灵活剪裁
  - 最小内核8KB
  - 共400多个模块
- 标准的开发接口
  - 部分支持ANSI C标准
  - 支持常见的各种IPC（进程间通信）机制
- 丰富的TCP/IP网络协议栈
- 支持多处理器（SMP）
- 提供丰富的调试工具

Workbench开发环境



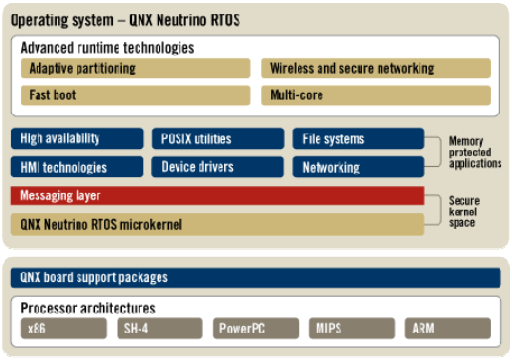
QNX嵌入式操作系统

- 加拿大的QNX公司推出，已被Research In Motion并购
- 特点：
  - 商用的类Unix实时操作系统，X86平台最好的RTOS
  - 非常小巧：4.x只有12k，运行速度极快
  - 典型微内核RTOS，核心仅提供4种服务
    - 进程调度，进程间通信，底层网络通信，中断处理
    - 内核独立运行在一个被保护的地址空间

QNX的结构特点

- QNX将应用程序和传统的操作系统功能隔离开，每个部分都有自己专有的地址空间
- 一个组成部分不能未经许可监测或破坏其他组成部分
- 应用程序的交互仅通过内核管理机制来进行，保证安全性和可控性

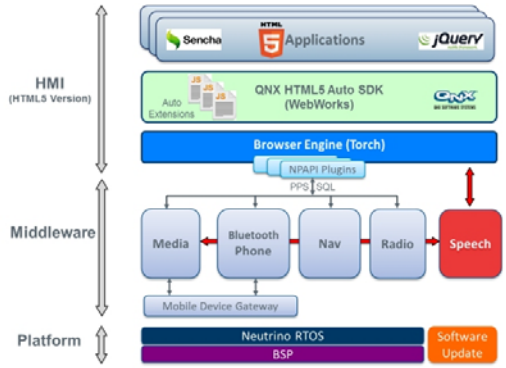
QNX架构



QNS微内核优点

- 驱动程序、网络协议、文件系统等一般OS模块与内核相互独立，任何一般OS模块的故障都不会导致内核崩溃
- 驱动程序、网络协议、文件系统和应用程序都处于程序空间，调用相同的内核API，开发和调试与应用程序没有分别
- 功能模块可以根据需要动态加载和卸载，不需要重新编译内核；
- 可以编写监视模块监控高可靠性模块，必要时可重新启动或加载模块而无需重启整个系统

QNX-Car



QNX-Car

Nucleus PLUS

- 95%的代码用ANSI C编写，便于移植到各种CPU
- 支持CPU类型最丰富的实时多任务操作系统。
- 以函数库的形式链接到目标应用程序中
- 内核在CISC中20kB, 在RISC中40kB, 1.5kB的数据结构空间
- 提供完全源代码，无产品版税
- 开发应用非常方便：无BSP开发，移植性强
- 基于Microsoft Developers Studio的嵌入式集成开发环境

Nucleus PLUS的特点

- 内核采用软件组件的方法
  - 每个组件具有单一明确的目的(C和汇编语言组成)，提供清晰的外部接口，不允许从外部对组件内的全局访问。
  - 各个组件非常易于替换和复用
    - 任务控制、内存管理、任务间通信、中断管理等16个组件
- 主要应用:网络设备，例如：路由器、机顶盒等。

μC/OS-II

- 经实际验证的可靠性
  - 被美国国防部批准用于航空、武器等领域
- 源代码公开（但不免费）
- 独特的收费模式
  - 非商业、教育使用免费
  - 商业使用，按产品或者产品线来收License费，而无需按每个卖出的产品再交Royalty
- 最早是作者(Jean J. Labrosse)个人的一个学术实现→商业化
  - 核心设计思想发表在1992年的Embedded System Programming杂志中
  - 目前是Micrium公司在维护

μC/OS-II的特点

- **移植性强**
  - 绝大多数源代码是ANSI C写的，只有和处理器硬件相关的那部分是汇编语言写的，这样非常便于移植。
- **实时性**
  - 内核具有可抢占式的实时多任务调度功能，全部函数调用与服务的执行时间都是可知的（或者说可确定的）。
- **开源＋短小精悍**
  - μC/OS-II仅仅是一个小的操作系统内核，仅包含简单的调度系统、内存管理和任务间通信机制。而嵌入式系统提供的常见功能，如文件系统、TCP/IP网络、图形界面都是通过额外的组件来提供。

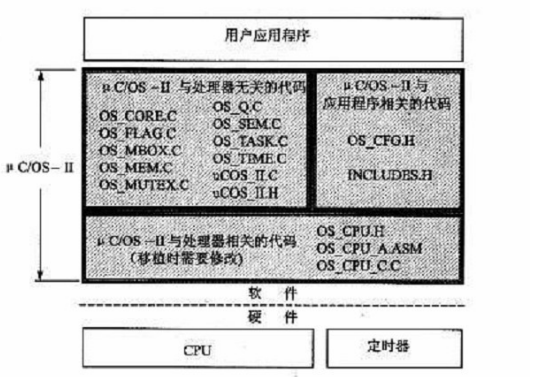
μC/OS-II的特点(续)

- **μC/OS-II体积小**
  - 具有可裁剪性，减少系统所需存储空间。
  - 极限情况下可压缩到只占用2KB的代码空间和200字节的数据空间！
- **与应用程序无严格界限**
  - 通常情况下，应用程序与操作系统会编译成一个独立的运行单元（μC/OS-II不能独立于应用程序单独运行）

μC/OS-III的特点

- **支持**时间片轮转法（**支持无限数量的任务**）
- 支持32位CPU，同时支持8/16位CPU
- 完备的内置性能测量功能
  - 能测量每一个任务的执行时间，每个任务的堆栈使用情况，任务的执行次数，CPU的使用情况， ISR 到任务的切换时间，任务到任务的切换时间，列表中的对象的峰值数，关中断、锁调度器平均时间等。
- 直接发送信号或消息到任务（不需要信号量）
- 任务可以同时等待多个内核对象
- ……

μCOS-II 的体系结构



Palm OS

- 3Com公司的产品，最初为掌上电脑设计
- 占用内存很小（通常几十KB）
- 主要特点
  - 节能：半休眠和休眠状态
  - 存储器全部是可读写的快速RAM：动态RAM和存储RAM（类似Flash）
  - 以数据库格式存储数据
  - 第一次提出与PC机进行数据同步的概念

嵌入式Linux

- **嵌入式Linux的由来**
  - 将通用Linux内核进行裁剪、定制修改后，应用到某些特定嵌入式领域里的Linux操作系统
- **优点**
  - 内核代码完全开放并免费。
    - 不同的应用可以根据自己的需求，对内核进行自由和深层次的定制修改。
  - 大量可用的第三方开源软件和开发工具
  - 符合IEEE POSIX标准，许多桌面应用程序可方便地移植到嵌入式Linux中

- 优点(续)
  - 功能更强大
    - 支持各种Internet网络协议
    - 支持ext2、fat、yaffs、romfs等最多种文件系统
  - 广泛的硬件支持
    - RISC or CISC，32位 or 64位，无MMU的处理器
- 缺点
  - 相对资源要求较高，一般要几个MB的ROM，十几MB的RAM
  - 缺乏一站式的文档和傻瓜式的开发工具，对开发者要求较高
  - 本身不是RTOS
    - 可以通过RT Linux方式解决

嵌入式Linux的不同版本

- RT-Linux(强实时)
  - 美国墨西哥理工学院开发，成功用于航天空间数据采集、科学仪器测控和电影特技图像处理等领域
  - 对Linux进行改造得到的版本
    - 通常的Linux任务低优先级，所有的实时任务高优先级

μCLinux

- 针对小型化应用祛除标准Linux的虚拟内存管理MMU部分代码（实地址访问）
- 特征：
  - 通用Linux API
  - 内核体积小于512k
  - 内核+文件系统小于900k
  - 支持完整的TCP/IP协议
  - 支持大量的网络协议和各种文件系统

嵌入式Linux应用例子

Nokia N810 Internet Tablet

Moto Razar2（第八版开始）

LinkSys WRT 54G无线路由器

松下Viera等离子彩电

Windows CE

- 专门设计的RTOS内核+类桌面Windows的GUI
  - 其内核并不是桌面Windows的简化版，而是完全从头开发的
- 最后版本 6.0/2007年
- 熟悉的Windows风格的GUI界面和操作习惯
- 丰富的类似Windows上的应用，如Outlook，Media player，Office
- 支持触屏操作
- 衍生产品 - 面向智能手机的Windows Mobile
- 主要市场 - PocketPC、GPS导航仪、智能手机等

基于Windows CE的上的Media Player

基于Windows CE的Windows Mobile 5界面

Windows CE特性

- 支持X86、ARM、MIPS、Super-H、……
- 一般需要几MB的RAM，几百KB的ROM（最小：200kb，加网络支持800kb，加图形界面支持4M，增加Internet支持3MB）
- 提供全部内核以及部分硬件相关模块的源代码
  - 但不是开源软件，需要支付Licence费和Royalty费
- 属于RTOS
  - 确定的中断响应延时
  - 在一个200MHz ARM上，约为10微秒之内
- 256个优先级的抢占式多任务
  - 使用优先级继承克服优先级反转的困难

Windows CE开发工具

- 系统移植 – Platform Builder
  - 提供BSP、Kernel的剪裁
- 应用开发
  - 提供类似桌面Windows的API
    - Windows程序员可以很快上手
  - Embedded Visual C++
  - 最新版的Visual Studio 2005

Windows CE 6.0的新特点

- 同时运行进程数32,000个,256个优先级
- 每个进程2GB的虚拟内存
- 免费提供Visual Studio.NET 2005
- 内核态与用户态的转变(胖内核)
- 100%共享源代码(相关开发工具和应用软件未公开)
- 强大的模拟器(可以模拟更多的CPU架构)
- ……

主要内容

- 嵌入式软件系统概况
- 嵌入式操作系统
- 常见的嵌入式操作系统
- 嵌入式软件开发和运行中的几个技术

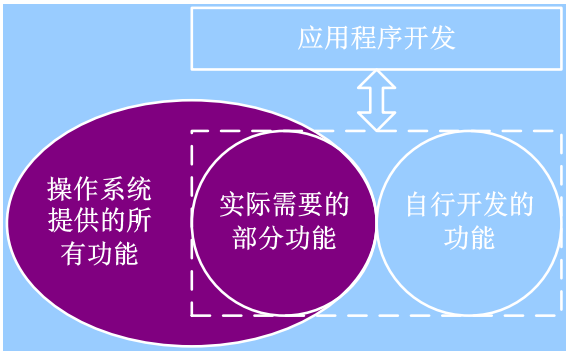
嵌入式软件运行和开发中的几个技术

- EOS的裁剪
- Bootloader引导（嵌入式linux类系统）
- BSP板级支持包（Windows CE系统）
- EOS移植

操作系统的定制裁剪

- 目的：减少操作系统所占的存储空间
  - 并非所有操作系统的功能在特定应用中都需要
  - 存储空间有限，不可能在发布时把冗余的系统功能都包含进去。
  - 许多商业嵌入式操作系统根据用户选择的组件来收取授权费用

操作系统的定制裁剪



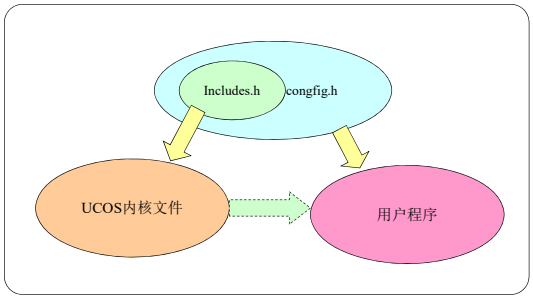
操作系统的定制手段

嵌入式操作系统	定制方式
Windows CE	提供Platform Builder集成开发环境，图形化组件选择。根据选择组件链接不同的lib文件
嵌入式Linux	内核部分在编译之前通过make config生成config配置文件，根据配置文件编译
Windows XP Embedded	提供Target Designer集成开发环境，图形化组件选择。根据选择的组件提取需要的二进制文件，不需要编译链接过程。
μC/OS-II	根据头文件中C语言宏定义的值，选择性条件编译部分代码，达到功能裁剪的目的。
VxWorks	提供Tornado集成开发环境，在Tornado中开发人员可以选择某些模块是否需要。

常见嵌入式操作系统的裁剪方式

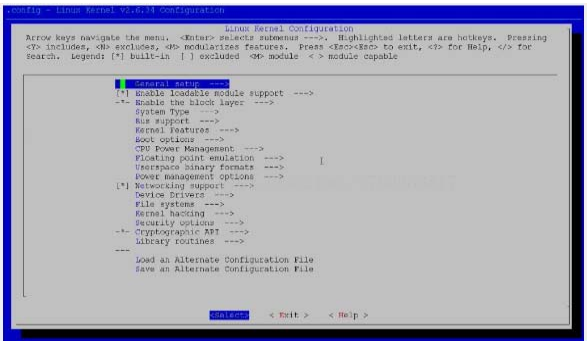
μC/ OS-II的剪裁实现

- 基于头文件的包含关系



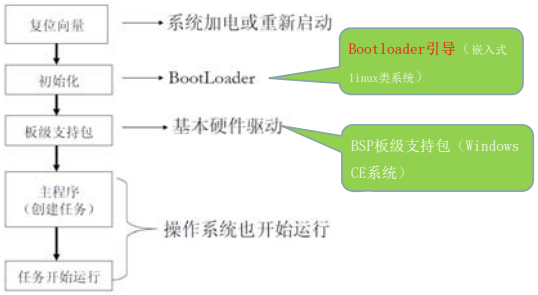
Windows CE
Platform Builder集成开发

LINUX内核配置menuconfig



嵌入式软件运行和开发中的几个技术

- 嵌入式软件运行过程



初始化引导技术

- 初始化引导代码是上电复位后第一个执行的代码功能：
  - 硬件初始化复位：
    - 80186：FFFF0H
    - MCS51：0x0000000
    - ARM：0x0000000
  - 执行操作：
    - 自检，基本硬件初始化
    - 引导操作系统
    - 进入初始化（硬件、软件）

Boot Loader引导技术

- 功能（类似BIOS）
  - 硬件初始化，引导和启动OS kernel。
- Boot Loader经常需要进行修改
  - 依赖于硬件：不同的嵌入式目标板即使基于同一种CPU构建，也需要修改Boot Loader源代码
    - 根据硬件具体配置修改参数，比如SDRAM的大小、Flash ROM的大小、使用的串口号、时钟频率等
    - 根据软件对CPU状态的要求：处于Supervisor模式，禁用IRQ和FIRQ，MMU关闭，D Cache关闭
    - ……

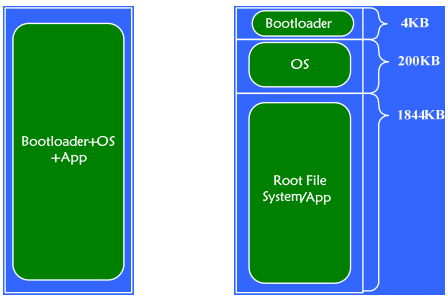
常见的Boot Loader

- 比较常见的Boot Loader如下：

Boot Loader	说明	X86	ARM	PPC	MIPS	SH	M68k
Lilo	早期的Linux磁盘引导加载程序	√					
GRUB	Lilo的GNU后续版，目前Linux默认的引导加载程序	√					
Blob	来自LART硬件计划的加载程序		√				
U-boot	以PPCBoot和ARMBoot为基础的通用加载程序		√	√			
Redboot	以eCos为基础的加载程序	√	√	√	√	√	√
Sh-Boot	Linux SH计划的重要加载程序					√	
vivi	韩国MIZI公司为Samsung S3C2410开发的加载程序		√				

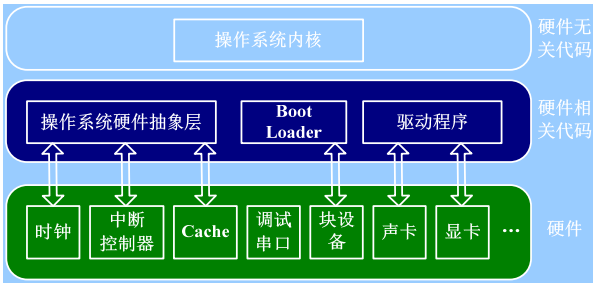
注：PPC是PowerPC处理器的缩写，SH是Super-H处理器的缩写

Boot loader的两种实现方式



单体式使用Flash ROM　　分区使用Flash ROM 的例子

BSP板级支持包



- 硬件抽象层与BSP（板级支持包，主要由Boot Loader和驱动程序构成）一起构成嵌入式系统层次结构中的中间层，使上层软件（操作系统）能独立于具体的硬件设备，它的引入大大推动了嵌入式操作系统的通用化。

嵌入式操作系统的移植

- 移植就是修改一个软件，使得它能够在新的环境下(不同的处理器平台，或者不同的处理器型号，或者不同的板子，或者不同的系统）下运行。
- 操作系统移植的任务：
  - 编写硬件抽象层HAL(Hardware Abstraction Layer)代码；
  - 定制裁剪EOS;

操作系统移植的内容

- 不考虑应用程序的移植，只考虑操作系统移植到不同平台上，主要的工作包括：
  - 理解处理器架构（寄存器，指令集，编程模式…)
  - 了解board的硬件设计（内存基地址？Flash地址？使用了处理器哪些资源？…)
  - 了解编译器（编译器传递参数的约定，支持内嵌汇编否？）
  - 理解操作系统的设计

UCOS-ii的移植

- UCOS-ii相对简单，移植工作量不大。
- 实现三个文件：

  os_cpu.h

  os_cpu_c.c

  os_cpu_a.s

UC/OS的移植

- 执行STI命令在Intel 80186上禁止中断

```
*********************************************************************************************************
*                                          与处理器相关的代码
*********************************************************************************************************
*/

#define  OS_ENTER_CRITICAL()  ???  /* 禁止中断                */                (3)
#define  OS_EXIT_CRITICAL()   ???  /* 允许中断                */

#define  OS_STK_GROWTH        1    /* 定义堆栈的增长方向:  1=向下，0=向上 */ (4)
```

```
#define OS_ENTER_CRITICAL()  asm CLI
#define OS_EXIT_CRITICAL()   asm STI
```

Os_cpu.h

- 主要定义通用的数据类型和堆栈增长方式，确定OS_ENTER/EXIT_CRITICAL的实现方式
- 数据类型是编译器相关的，堆栈增长方式是处理器相关的

```
typedef unsigned char  BOOLEAN;
typedef unsigned char  INT8U;             /* Unsigned  8 bit quantity                */
typedef signed   char  INT8S;             /* Signed    8 bit quantity                */
typedef unsigned int   INT16U;            /* Unsigned 16 bit quantity                */
typedef signed   int   INT16S;            /* Signed   16 bit quantity                */
typedef unsigned long  INT32U;            /* Unsigned 32 bit quantity                */
typedef signed   long  INT32S;            /* Signed   32 bit quantity                */
typedef float          FP32;              /* Single precision floating point         */

typedef unsigned char  OS_STK;            /* Each stack entry is 8- bit wide         */
typedef unsigned char  OS_CPU_SR;         /* Define size of CPU status register (PSW = 8 bits)  */

#define OS_CRITICAL_METHOD   3

#if     OS_CRITICAL_METHOD == 3
#define OS_ENTER_CRITICAL()  {cpu_sr = OS_CPU_SR_Save();}/* Disable interrupts                */
#define OS_EXIT_CRITICAL()   {OS_CPU_SR_Restore(cpu_sr);}/* Enable  interrupts                */
#endif

#define OS_STK_GROWTH    1                /* Stack grows from HIGH to LOW memory on AVR     */
#define OS_TASK_SW()     OSCtxSw()
```

Os_cpu_c.c

- 主要实现堆栈初始化函数，以及一些钩子函数（OSTimeTickHook，OSTaskSwHook …）
- 与编译器和处理器均相关

```
OS_STK *OSTaskStkInit (void (*task)(void *pd), void *p_arg, OS_STK *ptos, INT16U opt)
{
    INT8U *psoft_stk;
    INT8U *phard_stk;
    INT16U tmp;


    (void)opt;                          /* 'opt' is not used, prevent warning              */
    psoft_stk = (INT8U *)ptos;
    phard_stk = (INT8U *)ptos
              - OSTaskStkSize
              + OSTaskStkSizeHard;      /* Task stack size                                 */
                                        /* AVR return stack ("hardware stack")             */
    tmp       = *(INT16U const *)task;  /* (1) ICC compiler handles function pointers indirectly!  */

    *phard_stk-- = (INT8U)(tmp & 0xFF);  /* Put task start address on top of "hardware stack"  */
    tmp      >>= 8;
    *phard_stk-- = (INT8U)(tmp & 0xFF);

    *psoft_stk-- = (INT8U)0x00;          /* R0  = 0x00                                      */
    *psoft_stk-- = (INT8U)0x01;          /* R1  = 0x01                                      */
    *psoft_stk-- = (INT8U)0x02;          /* R2  = 0x02                                      */
    *psoft_stk-- = (INT8U)0x03;          /* R3  = 0x03                                      */
    *psoft_stk-- = (INT8U)0x04;          /* R4  = 0x04                                      */
    *psoft_stk-- = (INT8U)0x05;          /* R5  = 0x05                                      */
    *psoft_stk-- = (INT8U)0x06;          /* R6  = 0x06                                      */
    *psoft_stk-- = (INT8U)0x07;          /* R7  = 0x07                                      */
    *psoft_stk-- = (INT8U)0x08;          /* R8  = 0x08                                      */
```

Os_cpu.a.s

- 一个汇编语言文件，实现任务切换，现场保护/恢复
  - OSStartHighRdy
  - OSCtxSw
  - OSIntCtxSw
  - OS_CPU_SR_Save
  - OS_CPU_SR_Restore