嵌入式操作系统

武汉大学国家网络安全学院
丁玉龙 涂航

2024

内容提要

1	嵌入式操作系统简介
2	μC/OS
3	VxWorks
4	嵌入式Linux
5	COS与安全嵌入式操作系统

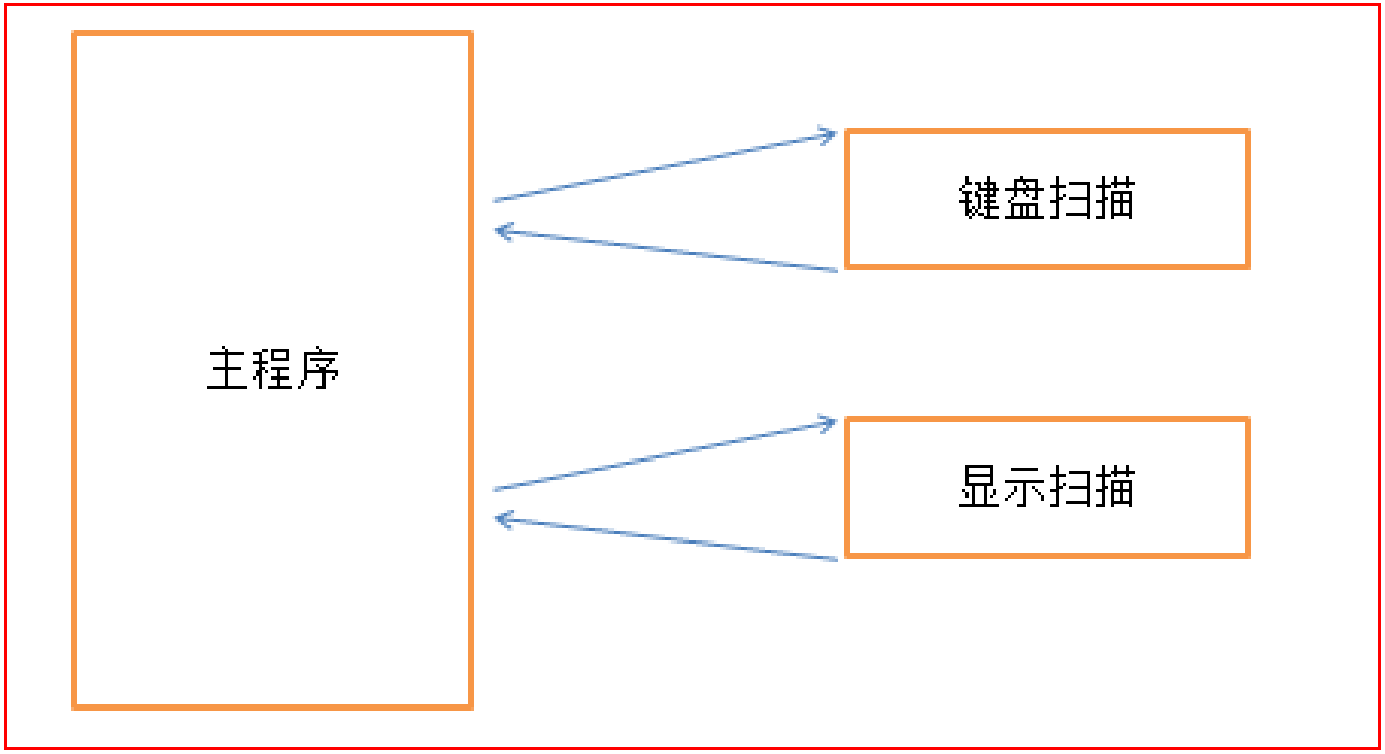
嵌入式软件设计的演变

- 顺序程序设计
- 基于状态机的程序设计
- 基于简易任务调度器的程序设计
- 基于操作系统的程序设计

顺序执行程序

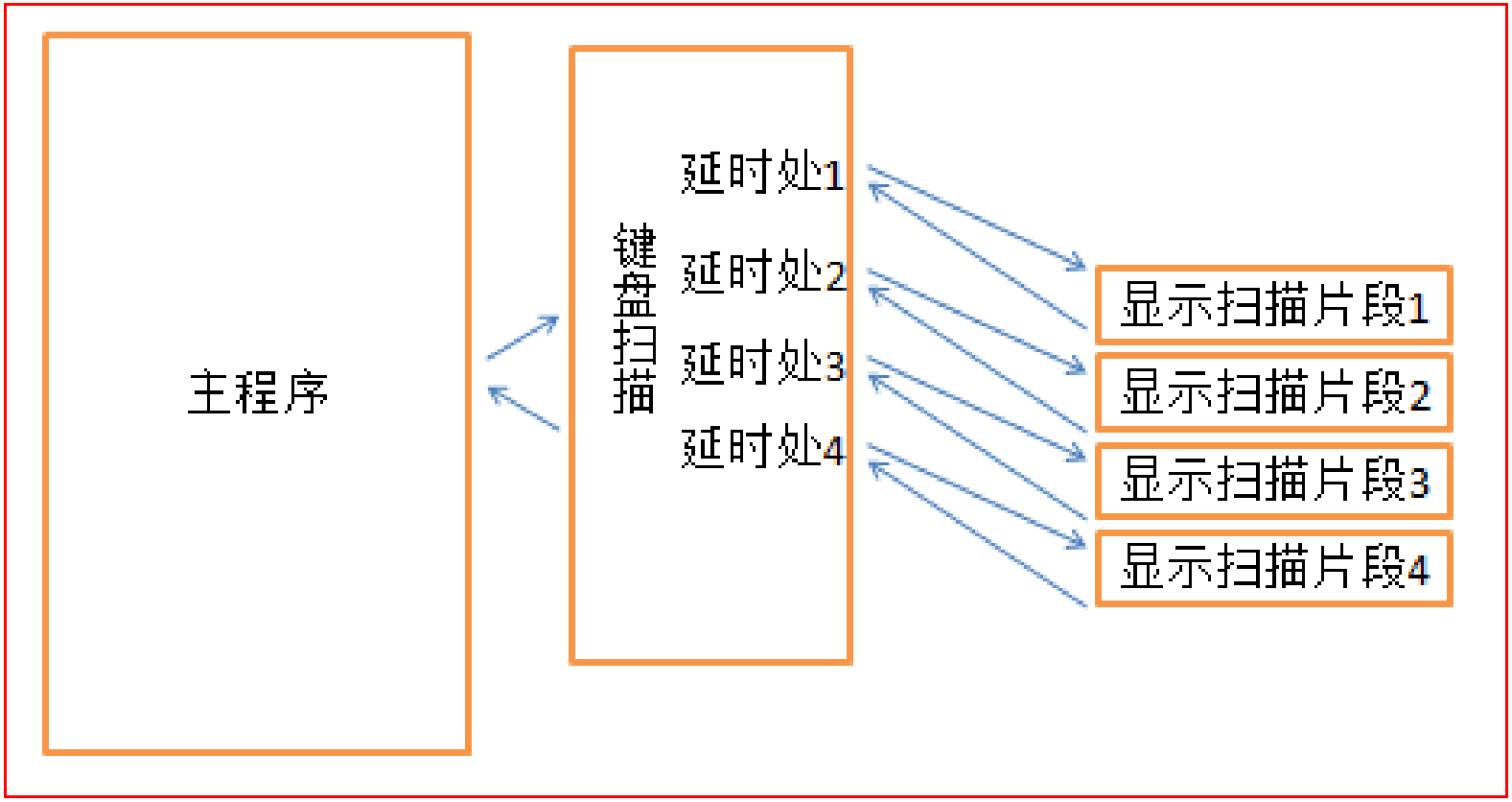
■顺序调用任务，执行完一个任务后再执行下一个任务。

■若任务长时间占用CPU，那么其它任务对外部事件的响应全部停止。



改进　将浪费的时间利用起来

■仔细观察可发现，其实任务并非一直运行，大部分时间是在延时。如果将任务**从延时处折断，分拆成小片段**后插入到另一个任务中，取代原有的延时程序，就可以提高系统资源的利用率。



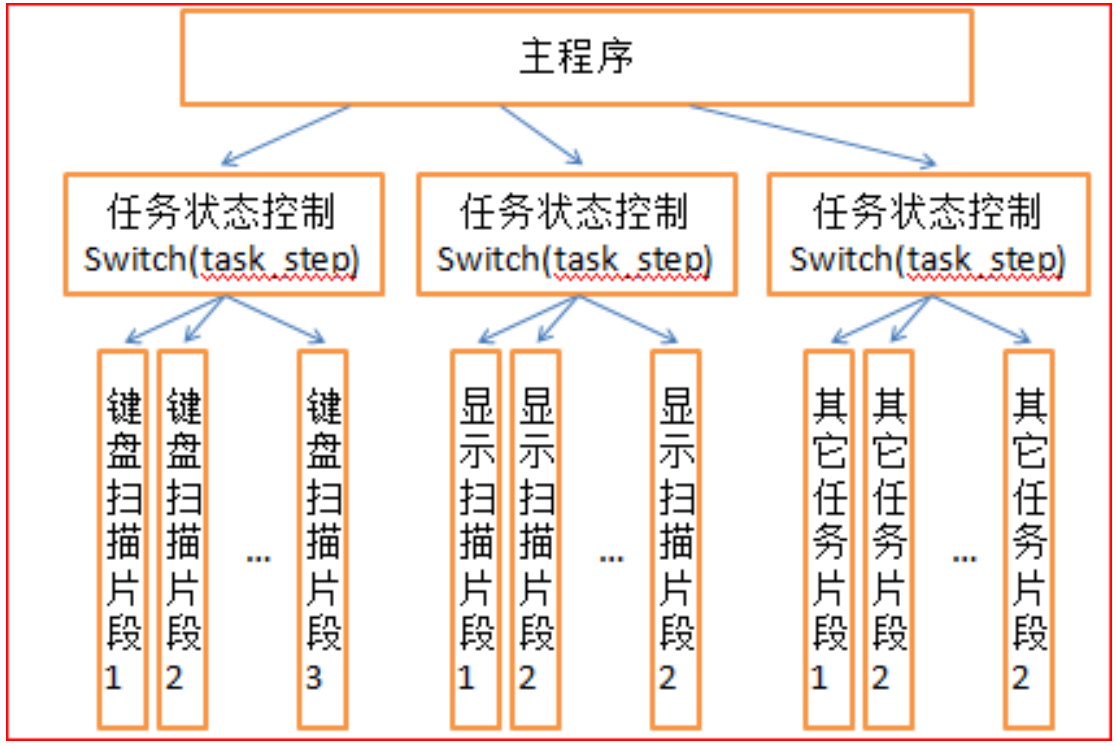
继续改进　实现流程控制---状态机

- 在主程序与任务之间增加一个接口:任务状态控制器。主程序只与任务的状态控制器打交道，由状态控制器负责调用任务的片段以及控制阶段的变换。

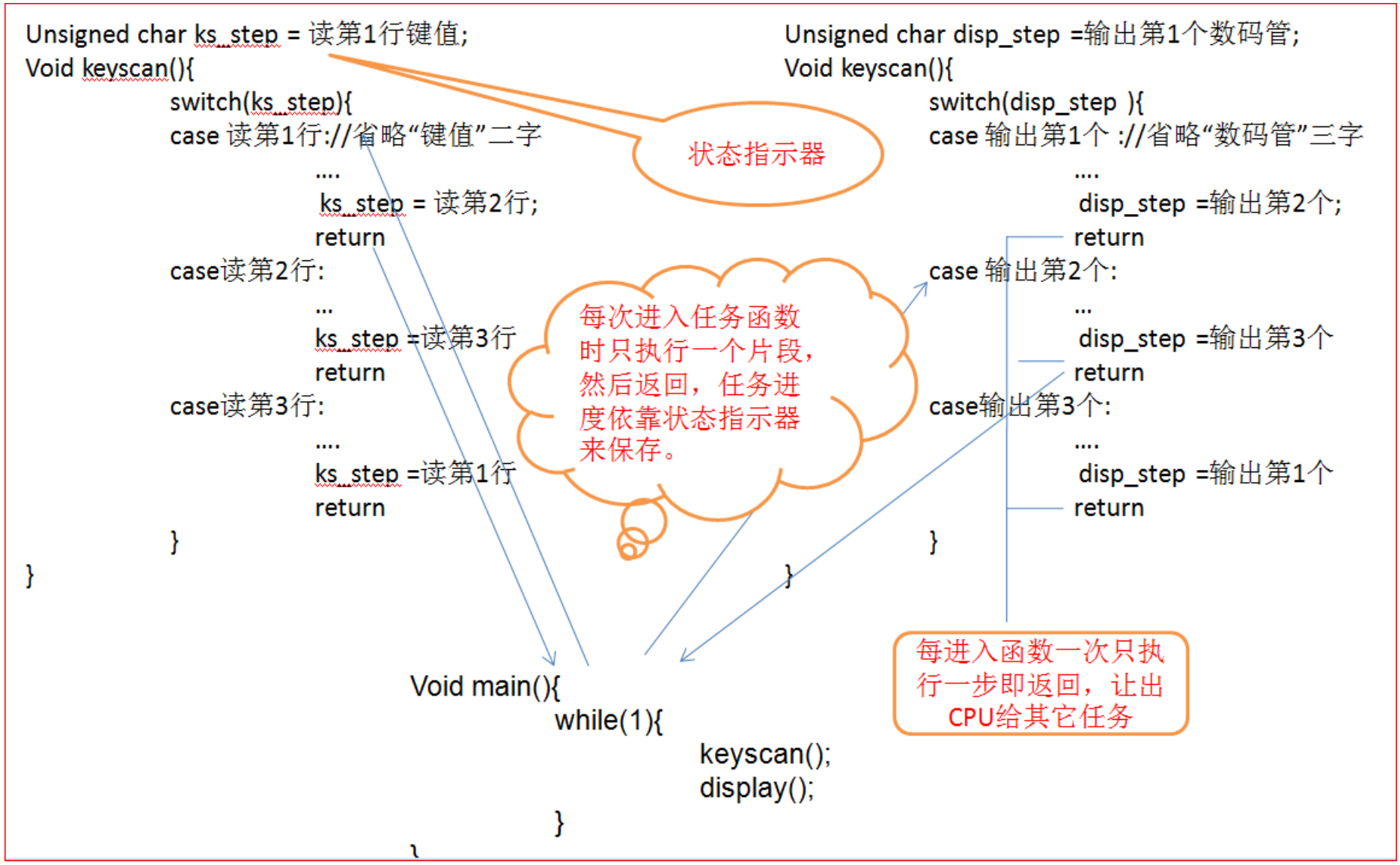
■状态机：

优点：占用资源少，执行效率高。

缺点：任务被拆得支离破碎，流程不直观。



状态机示例


Unsigned char ks_step = 读第1行键值;
Void keyscan(){
switch(ks_step){
case 读第1行://省略“键值”二字
....
ks_step = 读第2行;
return
case读第2行:
...
ks_step =读第3行
return
case读第3行:
....
ks_step =读第1行
return
}
}
状态指示器
每次进入任务函数时只执行一个片段，然后返回，任务进度依靠状态指示器来保存。
Unsigned char disp_step =输出第1个数码管;
Void keyscan(){
switch(disp_step){
case 输出第1个 ://省略“数码管”三字
....
disp_step =输出第2个;
return
case 输出第2个:
...
disp_step =输出第3个
return
case输出第3个:
....
disp_step =输出第1个
return
}
}
每进入函数一次只执行一步即返回，让出CPU给其它任务
Void main(){
while(1){
keyscan();
display();
}
}

实时系统概念

- 实时系统特点：如果逻辑和时序出现偏差将会引起严重后果的系统。
- 有两种类型的实时系统：软实时系统和硬实时系统。
- 软实时系统：各个任务运行得越快越好，并不要求限定某一任务必须在多长时间内完成。
- 硬实时系统：各任务不仅要执行无误而且要做到准时。
- 大多数实时系统是**二者的结合**。
- 实时应用软件的设计一般比非实时应用软件设计难。

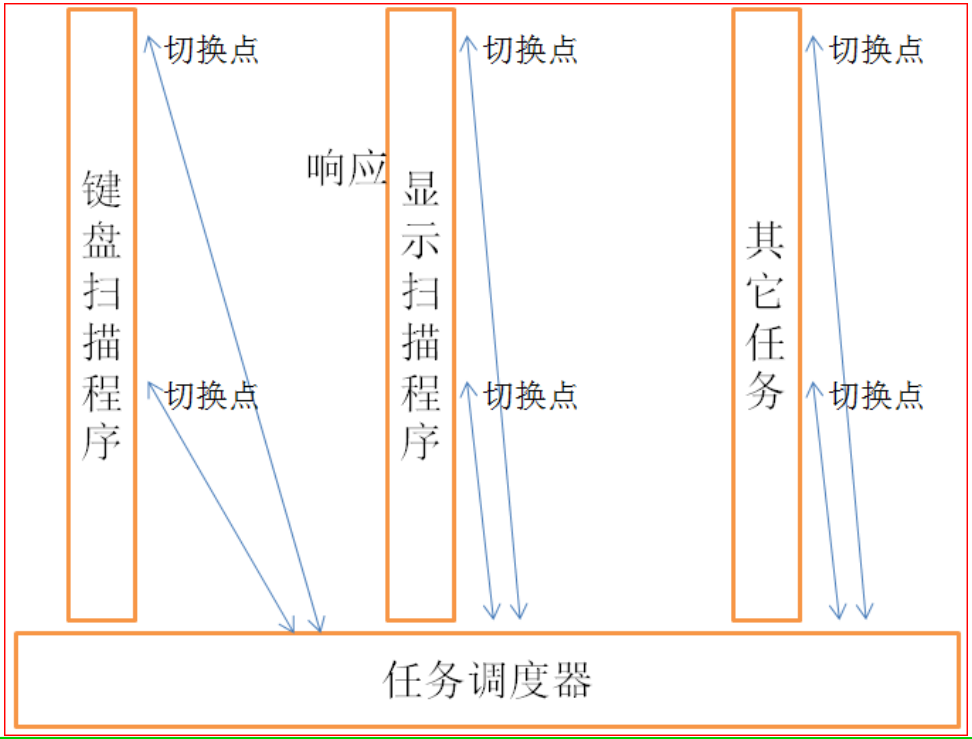
任务切换与任务调度

■**与顺序执行不同**的是:在执行完每个任务后，任务释放CPU，调度器分派下一个的任务接管CPU 。

■**与状态机不同的是**:状态机及主程序就是任务调用者，是主动调用者，任务片段是受调用者。而任务调度器中，任务调度者是被调用者。

■这种调用关系决定了任务又可以像以前顺序流程那样写成直观的任务函数。

■任务执行完任务片段后到达切换点，此时调用任务调度函数将该任务流折断，调度器将该任务堆栈保存后将CPU交给下一个任务。当该任务下次重新获行运行机会时,只需取出之前保存的堆栈,即可从切换点处继续运行。



任务调度器示例

```
Void keyscan(){
        while(1){
            outrow(row);
            getkey();
            task_switch();
            row++;
        }
}
```

```
Void display(){
        unsigned char cs = 0;
        while(1){
            out_cs(cs);
            out_data (dispbuf[cs]);
            task_switch();
            cs++;
        }
}
```

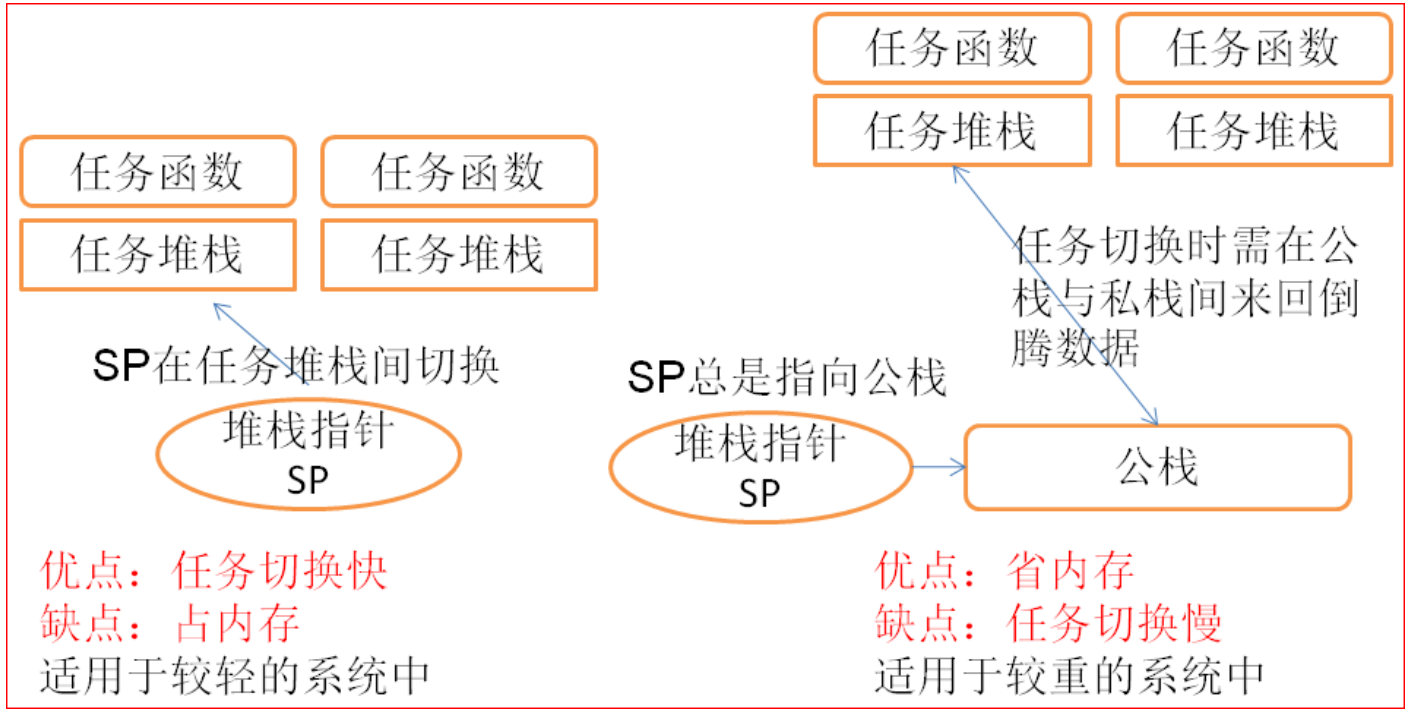
任务总是从切换处继续运行。流程可以自由拆分成任意片

```
Void task_switch(){
        保存当前任务的堆栈
        寻找下一个任务
        用下个任务的堆栈替换当前的堆栈
        return
}
```

同样是每次只执行一步即让出CPU，但任务函数比状态机优雅多了

任务调度的核心：堆栈迁移

- 有任务就有程序流，有程序流就需要堆栈，要折断任务流，就必须保存堆栈。每个任务都设有一个**私有任务堆栈**，用于保存任务被折断(任务切换)时的现场。
- 堆栈是上下文切换时最重要的切换对象，这种对堆栈的切换叫作“**堆栈迁移**”。
- **堆栈迁移有两种方式**，一种方法是(左图)使用私栈作为堆栈，发生任务切换时，只需将栈指针即可到新任务的栈顶即可。另一种是(右图)使用**公栈**作为作堆栈，每切换一个任务，就将公栈的内容搬向私栈，并将新任务从私栈搬至公栈，然后修改栈指针指向新的栈底



1 嵌入式操作系统简介

实时多任务操作系统与分时多任务操作系统

- 分时系统：软件的执行在时间上的要求并不严格，时间上的错误一般不会造成灾难性的后果。

- 实时系统：虽然事件可能在无法预知的时刻到达，但是软件上必须在事件发生时能够在严格的时限内作出响应（系统响应时间），即使是在尖峰负荷下，也应如此，系统时间响应的超时就意味着致命的失败。另外，实时操作系统的重要特点是具有系统的可确定性，即系统能对运行情况的最好和最坏等的情况能做出精确的估计。

1 嵌入式操作系统简介(续)

- 基本概念 ——时钟节拍

时钟节拍是特定的周期性中断。这个中断可以看作是系统心脏的脉动。周期取决于不同应用，一般在10ms到200ms之间。时钟的节拍式中断使得内核可以将任务延时若干个整数时钟节拍，以及当任务等待事件发生时，提供等待超时的依据。时钟节拍率越快，系统的额外开销就越大。

1 嵌入式操作系统简介(续)

实时操作系统中的重要概念

■系统响应时间(System response time)

系统发出处理要求到系统给出应答信号的时间。

■任务切换时间(Context-switching time)

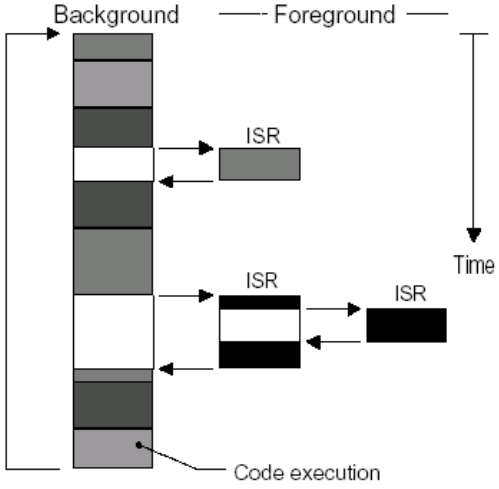
任务之间切换而使用的时间。

■中断延迟(Interrupt latency)

硬件接收到中断信号到操作系统作出响应，并转入中断服务程序的时间。

前后台系统 （Foreground/Background System）

- 前后台系统或超循环系统(Super-Loops)：应用程序是一个无限的循环，循环中调用相应的函数完成相应的操作，这部分可以看成后台行为(background)。中断服务程序处理异步事件，这部分可以看成前台行为（foreground）。后台也可以叫做任务级。前台也叫中断级。适用于不复杂的小系统。
- 时间相关性很强的关键操作靠中断服务来保证的。
- 前后台系统在处理信息的及时性上，比实际可以做到的要差。这个指标称作任务级响应时间。最坏情况下的任务级响应时间取决于整个循环的执行时间。因为循环的执行时间不是常数，程序经过某一特定部分的准确时间也是不能确定的。进而，如果程序修改了，循环的时序也会受到影响。



代码的临界段

- 代码的临界段也称为临界区，指处理时不可分割的代码。一旦这部分代码开始执行，则不允许任何中断打入。为确保临界段代码的执行，在进入临界段之前要关中断，而临界段代码执行完以后要立即开中断。
- 数据撕裂（例如大于CPU位宽的数据、大于CPU位宽的定时器访问）
- 临界段代码影响中断响应时间（因为要关中断）

资源

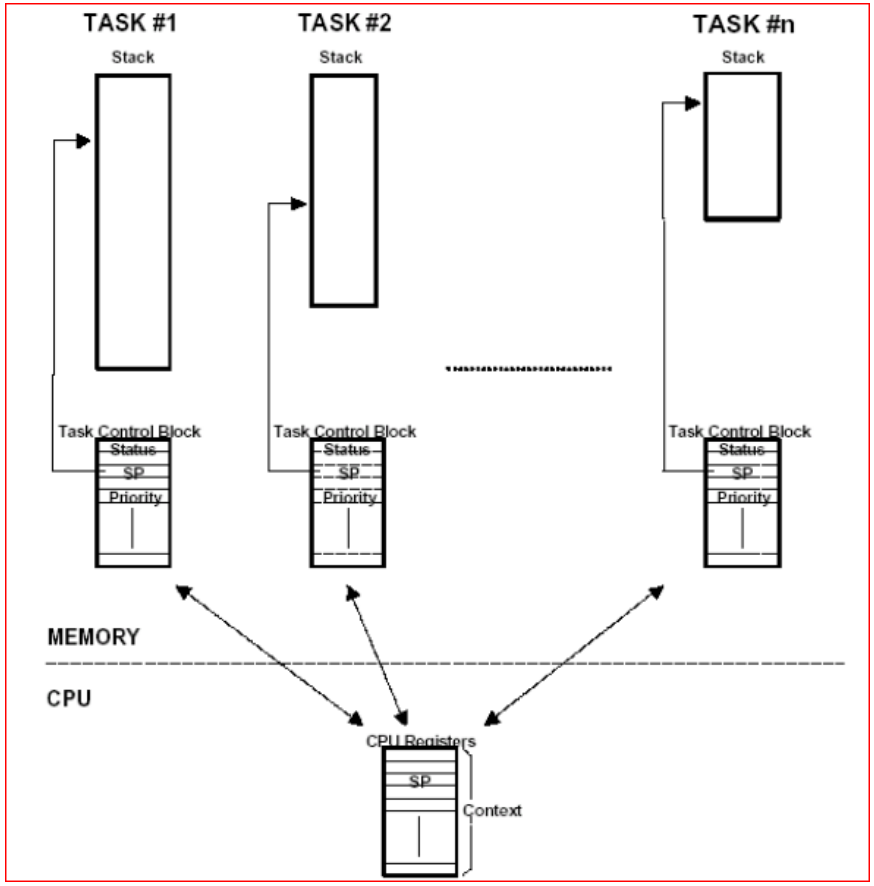
- 任何为任务所占用的实体都可称为资源。
- 资源可以是硬件设备，也可以是软件、存储空间。
- 例如打印机、键盘、显示器、函数、变量、结构体或数组等都可以是资源。

多任务

- 多任务运行的实现实际上是靠CPU(中央处理单元)在许多任务之间转换、调度。CPU只有一个，轮番服务于一系列任务中的某一个。多任务运行很像前后台系统，但后台任务有多个。多任务运行使CPU的利用率得到最大的发挥，并使应用程序模块化。
- 多任务化的最大特点是，开发人员可以将复杂的应用程序层次化。
- 使用多任务，应用程序将更容易设计与维护。

任务

- 任务，也称作一个线程，是一个简单的程序，该程序可以认为CPU完全只属于自己。实时应用程序的设计过程，包括如何把问题分割成多个任务，每个任务都是整个应用的某一部分，每个任务被赋予一定的优先级，有自己的CPU寄存器映射空间和栈空间。



任务状态

■5种状态

■休眠态：该任务驻留在内存中，但并不被多任务内核所调度。

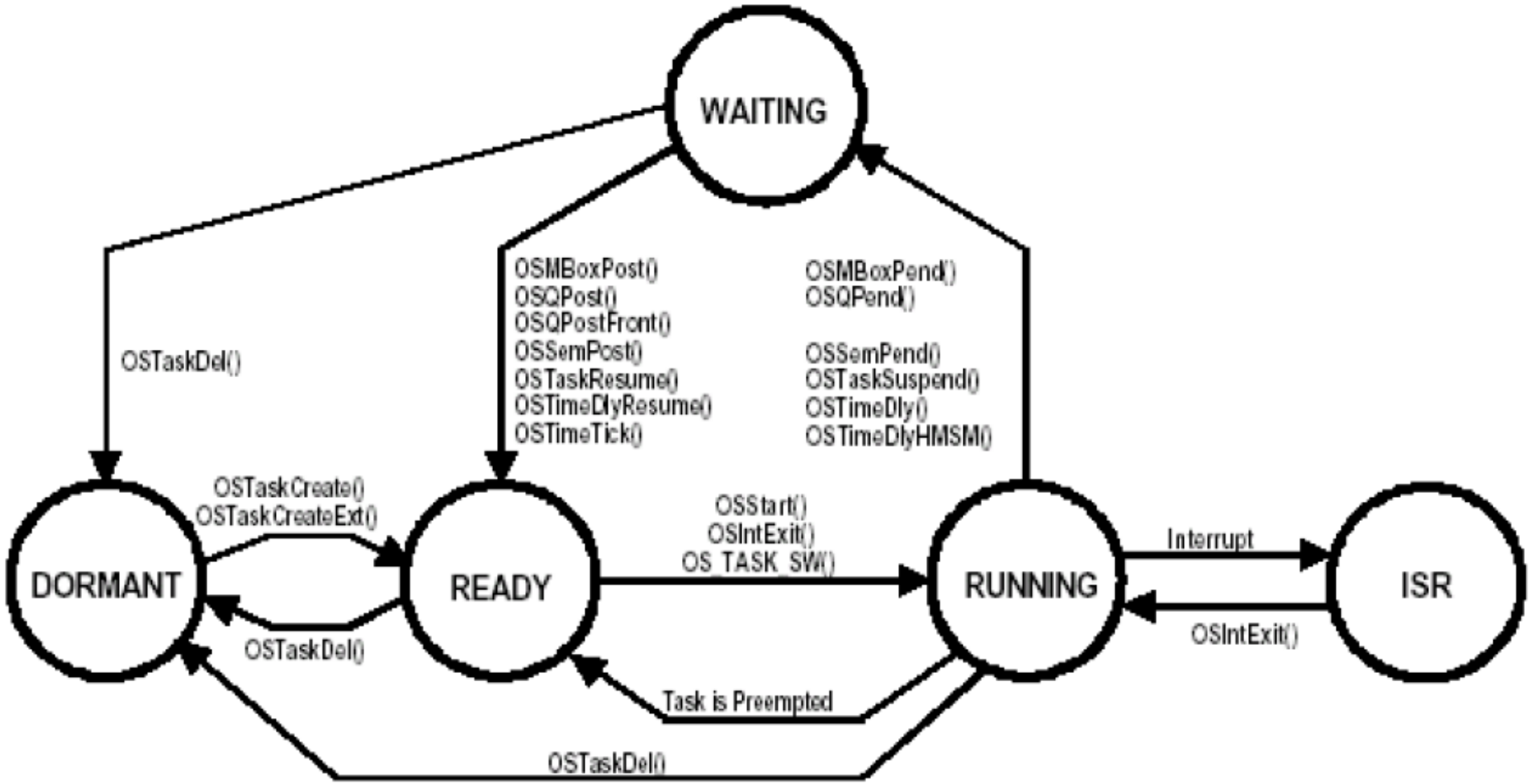
■就绪态：任务准备运行，但优先级比运行任务低，暂时不能运行。

■运行态：任务是指该任务掌握了CPU的控制权，正在运行中。

■挂起态：也可以叫做等待事件态。指该任务在等待某一事件的发生，（例如等待某外设的I/O操作，等待某共享资源由暂不能使用变成能使用状态，等待定时脉冲的到来或等待超时信号的到来以结束目前的等待，等等）。

■被中断态：发生中断时，CPU提供相应的中断服务，正在运行的任务暂不能运行，就进入了被中断状态。

任务状态


WAITING
OSMBoxPost()
OSQPost()
OSQPostFront()
OSSemPost()
OSTaskResume()
OSTimeDlyResume()
OSTimeTick()
OSMBoxPend()
OSQPend()
OSSemPend()
OSTaskSuspend()
OSTimeDly()
OSTimeDlyHMSM()
OSTaskDel()
OSTaskCreate()
OSTaskCreateExt()
DORMANT
READY
OSStart()
OSIntExit()
OS_TASK_SW()
RUNNING
Interrupt
ISR
OSIntExit()
OSTaskDel()
Task is Preempted
OSTaskDel()

任务切换(Context Switch or Task Switch)

- Context Switch 在有的书中翻译成上下文切换，实际含义是任务切换，或CPU寄存器内容切换。
- 当多任务内核决定运行另外的任务时，它保存正在运行任务的当前状态（Context），即CPU寄存器中的全部内容。这些内容保存在任务的当前状况保存区（Task’s Context Storage area），也就是任务自己的栈区之中。入栈工作完成以后，就是把下一个将要运行的任务的当前状况从该任务的栈中重新装入CPU的寄存器，并开始下一个任务的运行。这个过程叫做任务切换。
- 任务切换过程增加了应用程序的额外负荷。CPU的内部寄存器越多，额外负荷就越重。做任务切换所需要的时间取决于CPU有多少寄存器要入栈。实时内核的性能不应该以每秒钟能做多少次任务切换来评价。

内核（Kernel）

- 多任务系统中，内核负责管理任务，或者说为任务分配CPU时间，并且负责任务之间的通讯。
- 内核的基本服务是任务切换。之所以使用实时内核可以大大简化应用系统的设计，是因为实时内核允许将应用分成若干个任务，由实时内核来管理它们。
- 内核本身也增加了应用程序的额外负荷，代码空间增加ROM的用量，内核本身的数据结构增加了RAM的用量。但更主要的是，每个任务要有自己的栈空间（占内存来是相当厉害的）。
- 内核本身对CPU的占用时间一般在2-5%之间。
- 单片机一般不能运行实时内核，因为单片机的RAM很有限。
- 通过提供必不可缺少的系统服务，诸如信号量、邮箱、消息队列、延时等，实时内核使得CPU的利用更为有效。一旦用实时内核做过系统设计，将决不再想返回到前后台系统。

调度（Scheduler）

- 调度（Scheduler、dispatcher）。这是内核的主要职责之一，就是要决定该轮到哪个任务运行了。多数实时内核是基于优先级调度法的。每个任务根据其重要程度的不同被赋予一定的优先级。
- 基于优先级的调度法指，CPU总是让处在就绪态的优先级最高的任务先运行。
- 何时让高优先级任务掌握CPU的使用权，有两种不同的情况，这要看用的是什么类型的内核，是不可剥夺型的还是可剥夺型内核。

不可剥夺型内核 （Non-Preemptive Kernel）

- 不可剥夺型内核要求每个任务自我放弃CPU的所有权，也称作合作型多任务，各个任务彼此合作共享一个CPU。异步事件还是由中断服务来处理。中断服务以后控制权还是回到原来被中断的任务，直到该任务主动放弃CPU的使用权时，那个高优先级的任务才能获得CPU的使用权。
- 不可剥夺型内核的一个优点是响应中断快。不可剥夺型内核允许使用不可重入函数。因为每个任务要运行到完成时才释放CPU的控制权。任务级响应时间取决于最长的任务执行时间。
- 不可剥夺型内核的另一个优点是几乎不需要使用信号量保护共享数据。运行着的任务占有CPU，而不必担心被别的任务抢占。但这也不是绝对的。处理共享I/O设备时仍需要使用互斥型信号量。例如，在打印机的使用上，仍需要满足互斥条件。
- 不可剥夺型内核的最大缺陷在于其响应时间。高优先级的任务已经进入就绪态，但还不能运行，要等，也许要等很长时间，直到当前运行着的任务释放CPU。与前后系统一样， 不可剥夺型内核的任务级响应时间是不确定的。
- 商业软件几乎没有不可剥夺型内核。

可剥夺型内核

- 可剥夺型内核总是让就绪态的高优先级的任务先运行，中断服务程序可以抢占CPU，到中断服务完成时，内核让优先级最高的就绪任务运行（不一定是被中断了的任务）。任务级系统响应时间得到了最优化且是可知的。使用可剥夺型内核使得任务级响应时间得以最优化。
- 使用可剥夺型内核时，应用程序不应直接使用不可重入型函数。**调用不可重入型函数时，要满足互斥条件，**这一点可以用互斥型信号量来实现。如果调用不可重入型函数时，低优先级的任务CPU的使用权被高优先级任务剥夺，不可重入型函数中的数据有可能被破坏。

可重入性（Reentrancy）

- 可重入型函数可以被一个以上的任务调用，而不必担心数据的破坏。
- 可重入型函数任何时候都可以被中断，一段时间以后又可以运行，而相应数据不会丢失。
- 可重入型函数或者只使用局部变量，即变量保存在CPU寄存器中或堆栈中。如果使用全局变量，则要对全局变量予以保护。
- 应用程序中的不可重入函数引起的错误很可能在测试时发现不了，直到产品到了现场问题才出现。
- 处理多任务的新手，使用不可重入型函数时，千万要当心。

时间片轮番调度法

- 给任务分配优先级相当困难。
- 当两个或两个以上任务有同样优先级，内核允许一个任务运行事先确定的一段时间，叫做时间额度（quantum），然后切换给另一个任务。也叫做时间片调度。内核在满足以下条件时，把CPU控制权交给下一个任务就绪态的任务：
  - 当前任务已无事可做；
  - 当前任务在时间片还没结束时已经完成；
  - 当前任务时间片时间到。

任务优先级

- 每个任务都有其优先级。任务越重要，赋予的优先级应越高。
- 静态优先级：应用程序执行过程中诸任务优先级不变，则称之为静态优先级。在静态优先级系统中，诸任务以及它们的时间约束在程序编译时是已知的。
- 动态优先级 ：应用程序执行过程中，任务的优先级是可变的，则称之为动态优先级。实时内核应当避免出现优先级反转问题。
- 优先级反转 ：优先级反转问题是实时系统中出现最多的问题。为防止发生优先级反转，内核能自动变换任务的优先级，这叫做优先级继承(Priority inheritance) 。

任务优先级分配

- 给任务定优先级是复杂的事，因为实时系统相当复杂。许多系统中，并非所有的任务都至关重要。不重要的任务自然优先级可以低一些。实时系统大多综合了软实时和硬实时这两种需求。软实时系统只是要求任务执行得尽量快，并不要求在某一特定时间内完成。硬实时系统中，任务不但要执行无误，还要准时完成。
- 单调执行率调度法RMS(Rate Monotonic Scheduling),用于分配任务优先级。这种方法基于哪个任务执行的次数最频繁,执行最频繁的任务优先级最高。

互斥条件

■实现任务间通讯最简便到办法是使用共享数据。特别是当所有到任务都在一个单一地址空间下，能使用全程变量、指针、缓冲区、链表、循环缓冲区等，使用共享数据结构通讯就更为容易。虽然共享数据区法简化了任务间的信息交换，但是必须保证每个任务在处理共享数据时的排它性，以避免竞争和数据的破坏。

■与共享资源打交道时，使之满足互斥条件最一般的方法有：

1、关中断 ，关中断时间不能长，影响系统的中断响应时间

2、禁止做任务切换 ，给任务切换上锁和开锁

3、利用信号量

死锁（Deadlock (or Deadly Embrace)）

■死锁，指两个任务无限期地互相等待对方控制着的资源。设任务T1正独享资源R1，任务T2在独享资源R2，而此时T1又要独享R2，T2也要独享R1，于是哪个任务都没法继续执行了，发生了死锁。

■最简单的防止发生死锁的方法是让每个任务都：

1、先得到全部需要的资源再做下一步的工作；

2、用同样的顺序去申请多个资源；

3、释放资源时使用相反的顺序；

4、内核大多允许在申请信号量时定义等待超时，以化解死锁。

同步

- 信号量
- 互斥量
- 事件标志(Event Flags)，与多个事件同步

任务间的通讯(Intertask Communication)

- 任务间信息的传递有两个途径：通过全程变量或发消息给另一个任务。
- 消息邮箱(Message Mail boxes)
- 消息队列(Message Queue)

中断

- **中断响应时间** = 中断延迟 + 保存CPU内部寄存器的时间
- **中断处理时间**
- **中断恢复时间**(Interrupt Recovery)
- 中断恢复时间 = 判定是否有优先级更高的任务进入了就绪态的时间 + 恢复那个优先级更高任务的CPU内部寄存器的时间 + 执行中断返回指令的时间。
- 中断返回的处理：

前后台系统：程序回到后台程序；

不可剥夺型内核：程序回到被中断了的任务；

可剥夺型内核：就绪态的最高优先级任务运行；

中断延迟

- 实时内核的一个重要性能指标是中断最长要关多长时间。所有实时系统在进入临界区代码段之前都要关中断，执行完临界代码之后再开中断。关中断的时间越长，中断延迟就越长。
- 中断延迟 = 关中断的最长时间 + 开始执行中断服务子程序的第一条指令的时间

中断响应

- 中断响应定义为从中断发生到开始执行用户的中断服务子程序代码来处理这个中断的时间。中断响应时间包括开始处理这个中断前的全部开销。
- 中断响应是系统在最坏情况下的响应中断的时间。
- 前后台系统中断响应时间 = 中断延迟 + 保存CPU寄存器的时间；
- 不可剥夺型内核中断响应时间 = 中断延迟 + 保存CPU寄存器的时间；
- 可剥夺型内核中断响应 = 中断延迟 + 保存CPU寄存器的时间 + 内核的进入中断服务函数的执行时间 ；

中断恢复时间(Interrupt Recovery)

- 前后台系统中断恢复时间= 恢复CPU寄存器值时间 + 执行中断返回时间；
- 不可剥夺型内核中断恢复时间= 恢复CPU寄存器值时间 + 执行中断返回时间；
- 可剥夺型内核中断恢复时间= 判定是否有优先级更高的任务就绪态时间 + 恢复优先级更高任务CPU寄存器时间 + 执行中断返回时间。

中断处理时间

- 中断服务的处理时间应该尽可能的短，并没有绝对的限制。
- 在大多数情况下，中断服务子程序应识别中断来源，从中断源设备取得数据或状态，并通知真正做该事件处理的那个任务。
- 通知（通过信号量、邮箱或消息队列）是需要时间的。
- 如果事件处理需花的时间短于给一个任务发通知的时间，就应该考虑在中断服务子程序中做事件处理并在中断服务子程序中开中断，以允许优先级更高的中断打入并优先得到服务。

非屏蔽中断(NMI)

- 有时内核引起的延时变得不可忍受。在这种情况下可以使用非屏蔽中断，绝大多数微处理器有非屏蔽中断功能。通常非屏蔽中断留做紧急处理用，如断电时保存重要的信息。然而，如果应用程序没有这方面的要求，非屏蔽中断可用于时间要求最苛刻的中断服务。
- 非屏蔽中断的中断服务不能使用内核提供的服务，因为非屏蔽中断是关不掉的，故不能在非屏蔽中断处理中处理临界区代码。然而向非屏蔽中断传送参数或从非屏蔽中断获取参数还是可以进行的。参数的传递必须使用全程变量，全程变量的位数必须是一次读或写能完成的，即不应该是两个分离的字节，要两次读或写才能完成。

时钟节拍(Clock Tick)

- 时钟节拍是特定的周期性中断。这个中断可以看作是系统心脏的脉动。中断之间的时间间隔取决于不同的应用，一般在10mS到200mS之间。时钟的节拍式中断使得内核可以将任务延时若干个整数时钟节拍，以及当任务等待事件发生时，提供等待超时的依据。时钟节拍率越快，系统的额外开销就越大。
- 相对时间（Delay的抖动）
- 绝对时间 （定时任务）

对存储器的需求

- 前后台系统：对存储器容量的需求取决于应用程序代码。
- 多任务内核：总代码量 = 应用程序代码 + 内核代码（1K-100K）
- 每个任务需要单独的栈空间(RAM)。应用设计人员分配任务栈。
- 有些内核任务栈大小可分别定义；有些内核要求任务栈相同。
- 若内核不支持单独的中断栈：RAM总需求 = 应用程序RAM需求 + (任务栈需求 + 最大中断嵌套栈需求) * 任务数
- 若内核支持中断栈，RAM总需求 = 应用程序的RAM需求 + 内核数据区的RAM需求 + 各任务栈需求之总和 + 最多中断嵌套之栈
- 特别注意以下几点：

  定义函数和中断服务的局部变量；函数(即子程序)的嵌套 ；

  中断嵌套 ；库函数需要的栈空间 ；

内存管理

- 在ANSI C中可以用malloc()和free()两个函数动态地分配内存和释放内存。但是，在嵌入式实时操作系统中，多次这样做会把原来很大的一块连续内存区域，逐渐地分割成许多非常小而且彼此又不相邻的内存区域，也就是内存碎片。由于这些碎片的大量存在，使得程序到后来连非常小的内存也分配不到。
- 由于算法的原因，malloc()和free()函数执行时间是不确定的。
- 常用方法：固定大小分区、可变长度分区。

使用实时内核的优缺点

■使得实时应用程序的设计和扩展变得容易，不需要大的改动就可以增加新的功能。通过将应用程序分割成若干独立的任务，RTOS使得应用程序的设计过程大为减化。

■使用实时内核额外的需求是：内核的价格、额外的ROM/RAM开销、2到4百分点的CPU额外负荷、 使用硬件成本。

■RTOS价格可能包括：

RTOS的基本价格；

RTOS的开发座席费；

RTOS版权使用费；

RTOS软件年维护费。

什么时候该使用OS？

1. 当多个任务都很耗时，而这些任务可以并发调用，为了提高任务之间的调度并发性，应该考虑使用OS；
2. 当业务逻辑过于复杂，而通过自设计的调度器来调度时，使得设计不能简单，相较于OS往往趋于复杂，不易维护，为了使系统设计更加简单可靠，可以考虑使用OS；
3. 当系统资源充足，开发团队熟悉OS，使用OS更加节省开发时间；
4. 对于低功耗设备，使用前后台系统比使用OS方法更易控制系统的功耗，因为 OS系统方式，为了使任务调度及时，往往定时器中断调度更频繁，CPU的唤醒频率更频繁。

1 嵌入式操作系统简介

实时多任务操作系统与分时多任务操作系统

- 分时系统：软件的执行在时间上的要求并不严格，时间上的错误一般不会造成灾难性的后果。

- 实时系统：虽然事件可能在无法预知的时刻到达，但是软件上必须在事件发生时能够在严格的时限内作出响应（系统响应时间），即使是在尖峰负荷下，也应如此，系统时间响应的超时就意味着致命的失败。另外，实时操作系统的重要特点是具有系统的可确定性，即系统能对运行情况的最好和最坏等的情况能做出精确的估计。

1 嵌入式操作系统简介(续)

- 基本概念 ——时钟节拍

时钟节拍是特定的周期性中断。这个中断可以看作是系统心脏的脉动。周期取决于不同应用，一般在10ms到200ms之间。时钟的节拍式中断使得内核可以将任务延时若干个整数时钟节拍，以及当任务等待事件发生时，提供等待超时的依据。时钟节拍率越快，系统的额外开销就越大。

1 嵌入式操作系统简介(续)

实时操作系统中的重要概念

■系统响应时间(System response time)

系统发出处理要求到系统给出应答信号的时间。

■任务切换时间(Context-switching time)

任务之间切换而使用的时间。

■中断延迟(Interrupt latency)

硬件接收到中断信号到操作系统作出响应，并转入中断服务程序的时间。

1 嵌入式操作系统简介(续)

- **多处理器结构**
  - 实时应用的飞速发展，对嵌入式操作系统的性能提出了更高的要求。单处理器的嵌入式系统已不能很好地满足某些复杂实时应用系统的需要，开发支持多处理器结构的RTOS已成为发展方向
  - 这方面比较成功的系统有pSOSystem等

1 嵌入式操作系统简介(续)

Commercial RTOS Products

VxWorks	Wind RiverSystem	$40k/s & royalty	wrs.com
VRTX	Microtec	$20K/seat	
LynxOS	Bulue Cat Linux	$2.5k/seat	lynuxworks.com
Nucleus	Accelerated Tech.	$20K/s with source code	
CMX 8051Tools	Tasking	$1.29k	Tasking.com
RT/Studio IDE	Precise	$30K	psti.com
Embedded Linux	Green Hill	$7.9K Royalty free	ghs.com
Embedex Linux	Lineo	$5k/seat	lineo.com

1 嵌入式操作系统简介(续)

智能手机操作系统

- Symbian
  - 诺基亚为主，S60是主流
- Windows CE
  - 微软：“Windows Mobile 将熟悉的 Windows 体验扩展到了移动环境中
  - 现在更名为Windows Phone 7
- RIM
  - 加拿大BlackBerry（黑莓），邮件是特色
  - 对输入设备都做过特别设计，使输入设备可以频繁使用
- iPhone OS
  - Mac OS X的衍生，类Unix系统， Objective-C
- Palm OS
  - 3Com公司的产品，专用于掌上电脑，Palm OS节能、占有非常小的内存、内置数据库，封闭操作系统
- MeeGo
  - 英特尔和诺基亚宣布整合Moblin和Maemo
- Android
- MontaVista

内容提要

1	嵌入式操作系统简介
2	μC/OS
3	VxWorks
4	嵌入式Linux
5	COS与安全嵌入式操作系统

基本概念 ——前后台系统

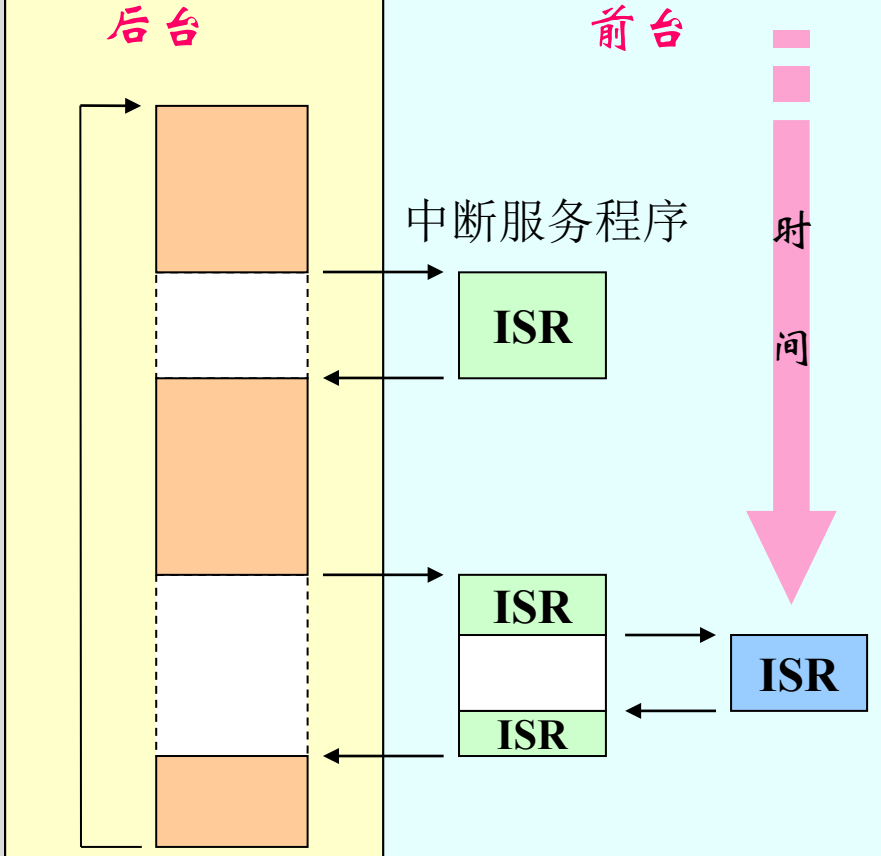
对基于芯片的开发来说，应用程序一般是一个无限的循环，可称为前后台系统或超循环系统。

很多基于微处理器的产品采用前后台系统设计，例如微波炉、电话机、玩具等。在另外一些基于微处理器应用中，从省电的角度出发，平时微处理器处在停机状态，所有事都靠中断服务来完成。

2 μC/OS (Cont.)

■ 基本概念 ——前后台系统

循环中调用相应的函数完成相应的操作，这部分可以看成后台行为，后台也可以叫做任务级。这种系统在处理的及时性上比实际可以做到的要差。



中断服务程序处理异步事件，这部分可以看成前台行为，前台也叫中断级。时间相关性很强的关键操作一定是靠中断服务程序来保证的。

2 μC/OS (Cont.)

μC/OS-II

μC/OS-II是一个源码公开、可移植、可固化、可裁剪、占先式的实时多任务操作系统。其绝大部分源码是用ANSI C写的，使其可以方便的移植并支持大多数类型的处理器。μC/OS-II通过了联邦航空局（FAA）商用航行器认证。自1992年问世以来，μC/OS-II已经被应用到数以百计的产品中。μC/OS-II占用很少的系统资源，并且在高校教学使用是不需要申请许可证。

2 μC/OS (Cont.)

- “μC/OS The Real Time Kernel” (1992)
- “MicroC/OS-II The Real-time Kernel” (1998)
- “MicroC/OS-II The Real-time Kernel” 2nd Edit (2002)

Jean J.Labrosse

R & D Publications, Inc www.cmpbooks.com

- Priority based preemptive kernel (looking up table algorithm)
- Up to 60 tasks
- Above 90% is written in C easy for porting
- Scalable and ROM able
- Source code for PC environment attached with a floppy or CD
- Very detail explanation in excellent programming style

内容提要

1	嵌入式操作系统简介
2	μC/OS
3	VxWorks
4	嵌入式Linux
5	COS与安全嵌入式操作系统

3 VxWorks

1）VxWorks概述

- 美国Wind River公司于1983年设计开发
- 高实时性和稳定性的微内核 、友好的用户开发环境 、良好的持续发展能力 ，全球商用市场占有量排名第一
- 广泛应用于通信、军事、航空、航天等实时性要求高领域
  - 美国F-16、 FA-18战斗机、B-2 隐形轰炸机和爱国者导弹上
  - 1997年和2004年两次在火星表面登陆的火星探测器
  - 商业用户包括Cisco systems、Bay Networks、3Com、Fore systems、HP、Lucent、Qualcomm、以及国内的华为、东方电子等

3 VxWorks (Cont.)

2) VxWorks基本特征

- (1) 高实时性、高稳定性的微内核
  - 内核Wind：微内核结构，最小8KB
  - 实时性
    - 基于优先级的抢占式调度辅以时间片轮转

      及时响应高优先级的任务，同级任务可选择时间片轮转而并发执行
    - 快速的任务上下文切换

3 VxWorks (Cont.)

- 较小的中断延时

  相应措施如：

  - 采用中断处理与任务在不同栈中处理，使得中断的产生只会引发一些关键寄存器的存储而不会导致任务的上下文切换
  - 在中断服务程序中只完成在最小时间内中断发生通告，而将其它费时的处理过程尽量放在被引发的其它任务中完成

- 高稳定性

3 VxWorks (Cont.)

- (2) **丰富的外挂组件模块**
  - 基本外挂组件模块

    各种设备驱动（字符型/块型设备，同步/异步设备）、文件系统（如DosFs、RawFs、TapeFs、CdromFs、TSFS等）、网络协议栈、以及POSIX1003.1b标准和ANSI C等兼容组件模块
  - 附加组件模块

    如Flash文件系统、图形界面管理等

3 VxWorks (Cont.)

- (3) **可裁减性**
  - 粒度极小的配置裁减性能
    - 微内核结构，最简内核（只负责任务的管理与调度，称为纳核）只有8KB
    - 其它所有基本外挂组件和附加组件均为可选组件，并且这些组件本身也是可裁减的
  - 方便友好的配置裁减环境
    - 图形化
    - 自动裁减特性，自动分析功能

3 VxWorks (Cont.)

- (4) 对多种硬件平台的可移植性
  - 支持ARM、PowerPC、68K、CPU32、SPARC、i960、X86、MIPS等众多嵌入式处理器，并提供相应的BSP模板
  - 提供了数量众多的串口、并口、网口、存储卡控制器、实时时钟等外围硬件设备的驱动程序

3 VxWorks (Cont.)

- (5) 友好、开放的集成开发环境
  - Tornado的可视化图形操作界面，可运行在多种主机硬件平台和操作系统上
    - 支持的主机硬件平台：Sun、HP、IBM-rs6000、Mips等
    - 支持的主机操作系统：Unix、WindowsNT/95/98等
  - Tornado的IDE

    集成了编辑器、编译器、链接器、调试器（命令行和图形界面两类调试器）、软件仿真器、工程项目管理器等系列开发工具
  - Tornado的开放性

    能与第三方开发工具进行集成

内容提要

1	嵌入式操作系统简介
2	μC/OS
3	VxWorks
4	嵌入式Linux
5	COS与安全嵌入式操作系统

4 嵌入式Linux

Linux的版本

- 内核：属于单内核
- 内核版本：最新的版本2.6.39
- 发行版
  - 基于Debian，如Ubuntu
  - 基于RPM，如Fedora，红旗
  - Slackware
  - 其它打包方式的套件
  - 专用包：Android/Maemo/Moblin/MeeGo
    - 2010年2月，谷歌Android被Linux内核除名
- 知识产权
  - 需要遵守GPL GNU Public License, LGPL: Lesser GPL
  - 没有独立的知识产权

4 嵌入式Linux（续）

- 典型的嵌入式Linux
  - Linux: Embedix，ETLinux，LEM，RTLinux，LinuxRouterProject，LOAF，uCLinux，muLinux，ThinLinux，FirePlug和PizzaBoxLinux
  - 平台化嵌入式Linux
    - Android
    - Moblin+Maemo→MeeGO

4 嵌入式Linux（续）

嵌入式Linux种类

是否支持MMU

MMU 实现虚拟存储空间：即将虚拟存储空间影射到实际物理存储空间。使编程人员不用考虑具体程序所放在物理存储空间的具体位置和程序的大小。

存储器访问权限的控制：任务间通讯，对自己的内存、堆栈等进行保护，只能通过管道、信号量、共享内存等方式进行通讯。

控制Cache

不支持MMU：ucLinux，主要应用在ARM7系列微处理系统中，如三星公司的S3C44B0，S3C4510等，无MMU，不支持虚地址，直接访问内存，所有程序中访问的地址都是物理地址。

支持MMU：嵌入式Linux，主要应用在ARM9系列微处理器系统中，如三星公司的S3C2410，Intel公司的PXA255等

4 嵌入式Linux（续）

- RT-Linux　美国新墨西哥州大学计算机系研制开发的

RT-Linux是利用Linux进行实时系统开发比较早的尝试，是一种硬实时操作系统。目前RT-Linux已成功应用于航天飞机的空间数据采集、科学仪器测控，以及电影特技图像处理等众多领域。

RT-Linux的原理是采用双内核机构，即将Linux的内核代码进行少量修改，将Linux任务以及Linux内核本身作为实时内核的一个优先级最低的任务，即实时任务优先级高于普通Linux任务，即在实时任务存在的情况下运行实时任务，否则才运行Linux本身的任务。实时任务不同于Linux普通进程，它是以Linux的内核模块(Linux Loadable KernelModule，LKM)的形式存在的。需要运行实时任务的时候，将这个实时任务的内核模块插入到内核中去。实时任务和Linux一般进程之间的通信通过共享内存或者FIFO通道来实现。

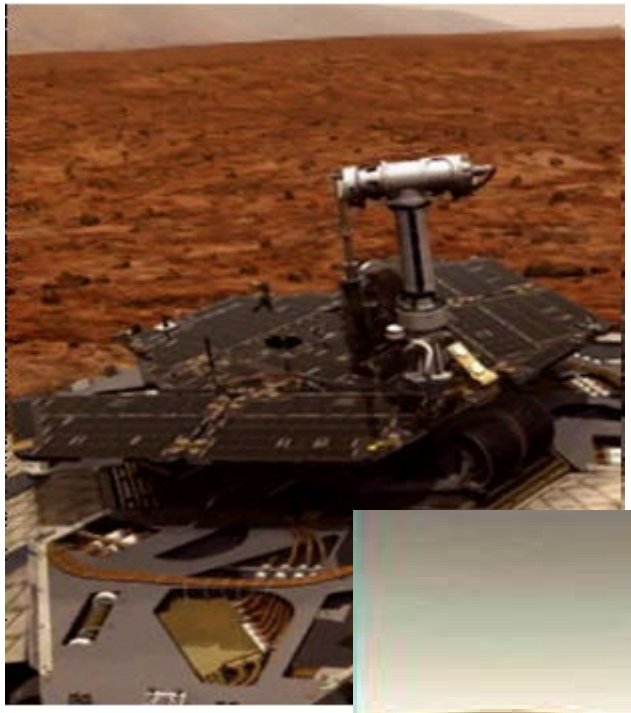
4 嵌入式Linux（续）

基于RTLinux的仿人机器人

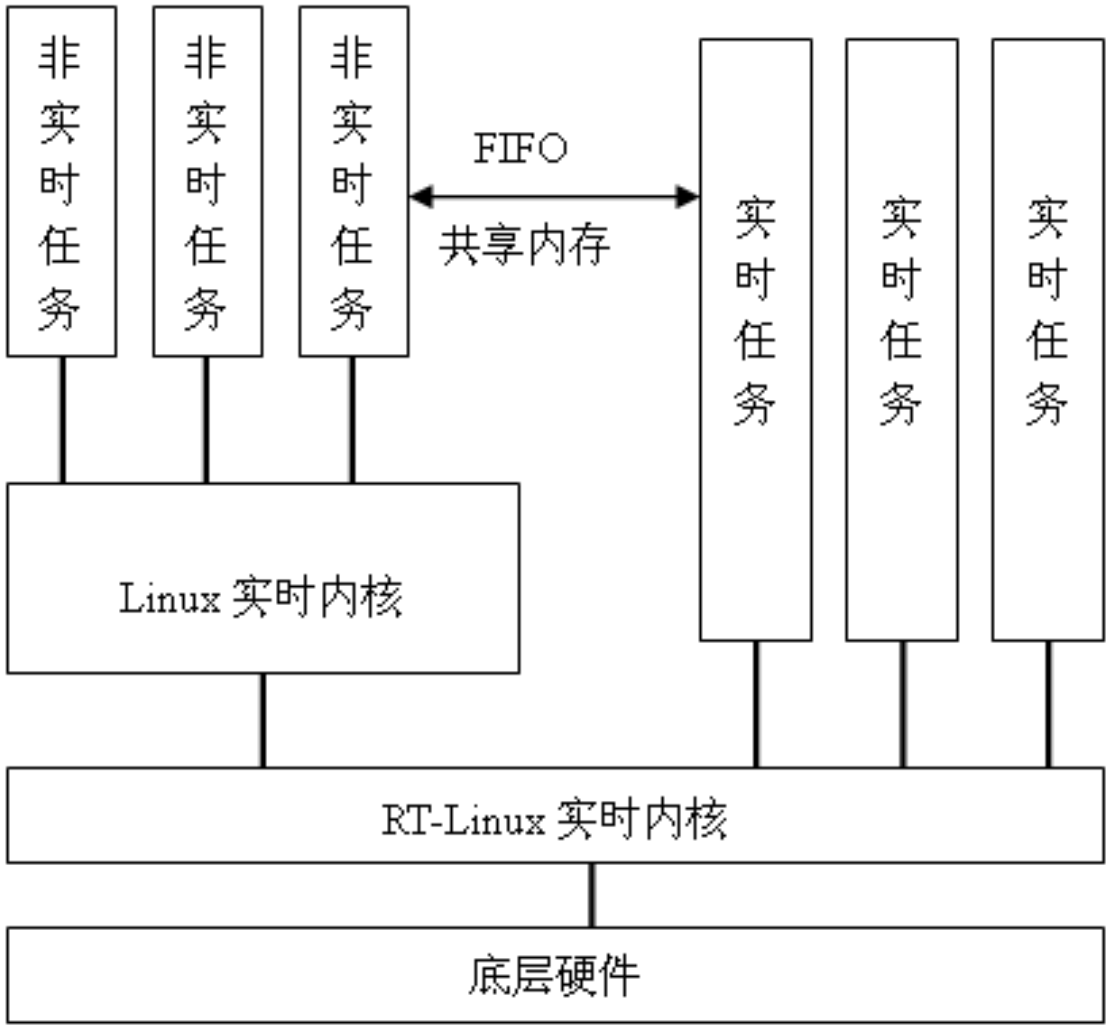
高 48 cm
重：6 kg
灵活性：20 DOF
操作系统：RT-Linux
接口形式：USB 1.0 (12Mbps)
响应周期：1ms
能源： DC24V x 6.2A (150W)
制造：富士通

4 嵌入式Linux（续）

2004年“勇气号”再次登陆火星



RT-Linux的工作原理图



4 嵌入式Linux（续）

经过加入实时处理后，RT Linux就完全能够达到硬实时系统的性能指标。在一台386机器上，RT Linux从处理器检测到中断，再到中断处理程序开始工作不会超过15μs；对一个周期性的任务，在*35μs*内一定会执行。

而通常的Linux内核，一般是在*600μs*内开始一个中断服务程序。

4 嵌入式Linux（续）

uClinux

- 控制领域中的linux系统。
- 它包含linux常用的API，内核小于512K，保留了linux原有的高稳定性、强大的网络功能和卓越的文件系统支持功能等优点。目前已支持的CPU芯片有，Motorola公司的68K系列、PowerPC系列以及ARM公司的系列芯片。
- uClinux最大特点就是不支持MMU。 uClinux系统对内存的访问是直接的，即不需要经过MMU，直接将地址发送到地址线上，所有程序访问的都是实际的物理地址，这样一方面减小了内核的体积，另一方面又增强了系统的实时性能。但内存空间得不到保护，对于应用开发者来说，必须明白自己程序运行的位置，以及保证不会破坏其它程序运行空间以及系统的稳定。
- uClinux也可以使用RT-Linux的实时补丁，以增强其实时性。

4 嵌入式Linux（续）

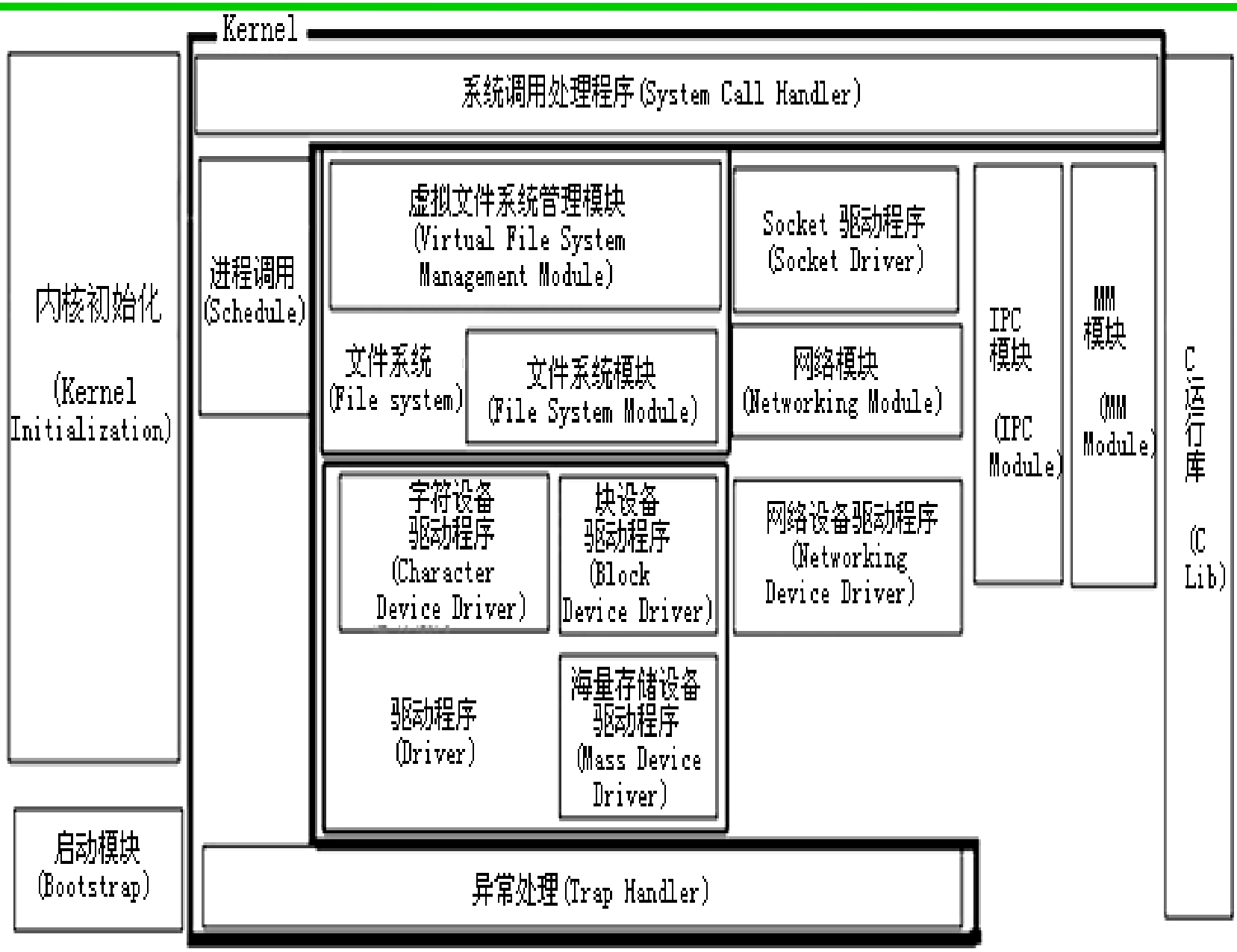
针对这种没有MMU的CPU架构，uClinux采用了一种平板式(Flat)的内存模型来去除对MMU的依赖，并且改变了用户程序的加载方式，开发了运用于uClinux的C函数库--uCLibc.

目前,**uClinux**往往基于两个Linux内核版本,*2.0.38*是一个比较成熟的版本,*2.4.x*是最新的版本。 Hitool 套件同时提供了对他们的支持。

一般uClinux的内核大小在500k左右,如果加上一些基本的应用,也就在900k左右.非常适合于嵌入式系统。

uClinux架构如下图所示

4 嵌入式Linux（续）


Kernel
系统调用处理程序(System Call Handler)
内核初始化
(Kernel Initialization)
进程调用
(Schedule)
虚拟文件系统管理模块
(Virtual File System Management Module)
Socket 驱动程序
(Socket Driver)
IPC
模块
(IPC Module)
MM
模块
(MM Module)
C语言库
(C Lib)
文件系统
(File system)
文件系统模块
(File System Module)
网络模块
(Networking Module)
字符设备
驱动程序
(Character Device Driver)
块设备
驱动程序
(Block Device Driver)
网络设备驱动程序
(Networking Device Driver)
驱动程序
(Driver)
海量存储设备
驱动程序
(Mass Device Driver)
启动模块
(Bootstrap)
异常处理(Trap Handler)

4 嵌入式Linux（续）

嵌入式Linux的组成

最基本的嵌入式Linux系统需要3个基本元素：

系统引导程序：用于完成机器加电后的系统定位引导；

Linux系统内核：为嵌入式应用提供一个软件环境，为应用程序完成基本的底层的资源管理工作；

初始化过程：完成基本的初始化。

4 嵌入式Linux（续）

为使这个最小嵌入式系统具有一定的实用性，还需加上硬件的驱动程序及一个或几个应用进程以提供必要的应用功能支持。

如果应用比较复杂，可能还需要添加一个可以在ROM或RAM中使用的文件系统、TCP/IP网络协议栈等。

在手机等领域，还需要加上一个GUI

4 嵌入式Linux（续）

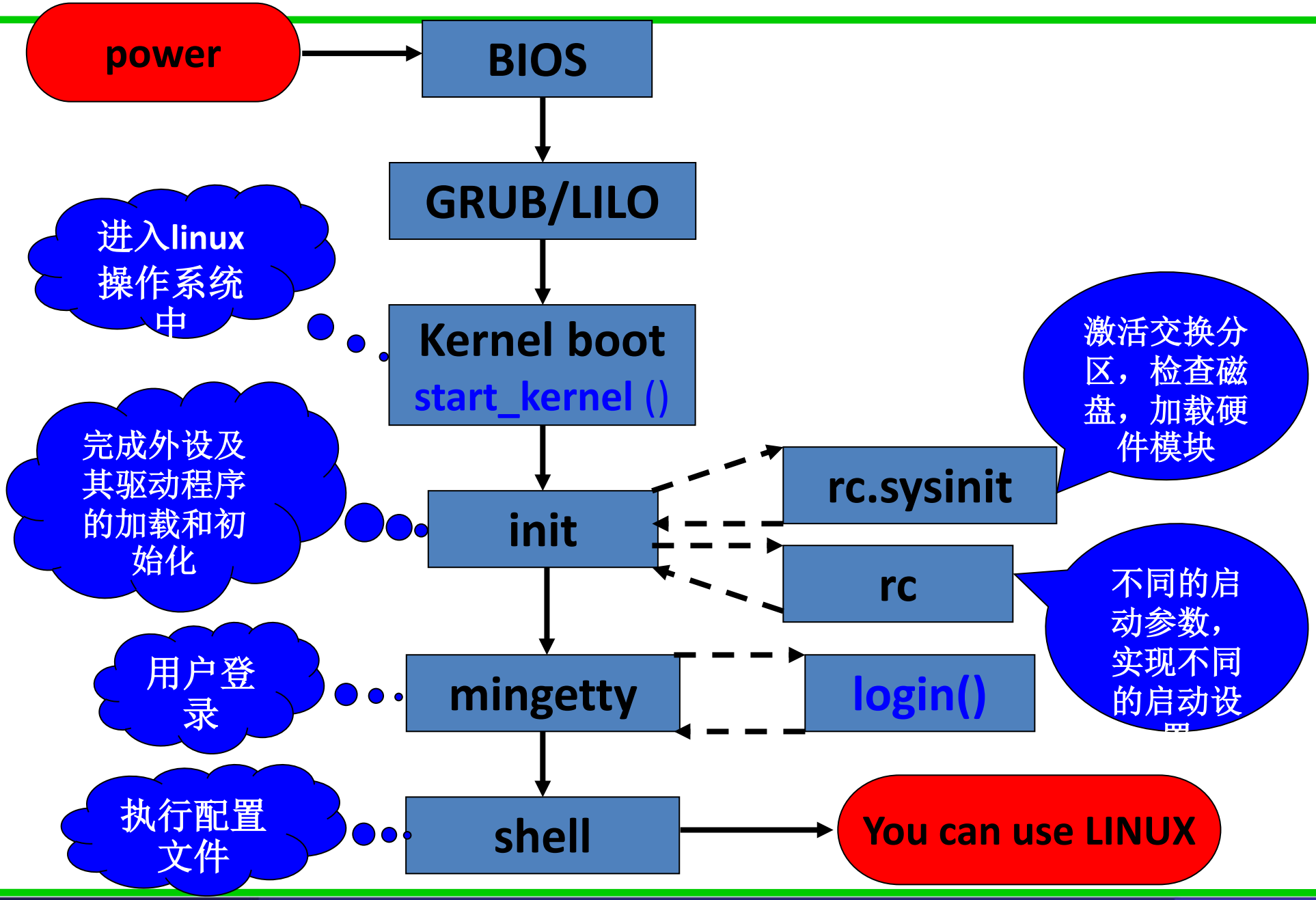
Linux启动

- linux的启动

  在x86体系中，cpu上电后在存储位置0xffff0处开始执行程序代码，这个地址通常是ROM-BIOS中的地址。

  - BIOS
    - 硬件检测，资源分配。
    - 将boot loader加载到RAM，然后将控制权交给boot loader。
  - boot loader
    - 将内核映像从硬盘中加载到RAM中，然后跳到内核的入口点，启动操作系统。

4 嵌入式Linux（续）

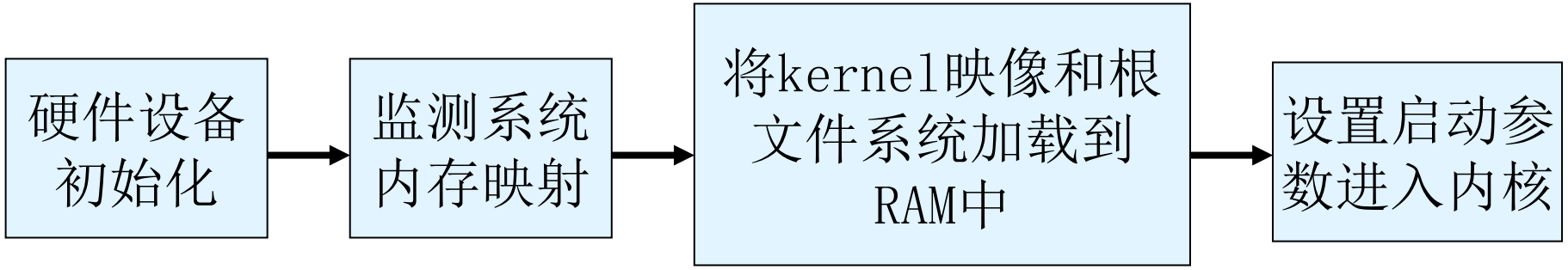

power
BIOS
GRUB/LILO
进入linux操作系统中
Kernel boot
start_kernel ()
完成外设及其驱动程序的加载和初始化
init
激活交换分区，检查磁盘，加载硬件模块
rc.sysinit
rc
不同的启动参数，实现不同的启动设
用户登录
mingetty
login()
执行配置文件
shell
You can use LINUX

4 嵌入式Linux（续）

嵌入式linux bootloader

- Boot loader的结构
  - stage1（汇编语言实现）

  - stage2（c语言实现）



4 嵌入式Linux（续）

嵌入式系统的存储

在嵌入式系统中使用Flash存储器

文件系统类型:

ext2:专用于Linux,引导块、超级块、inode、数据区。

cramfs:压缩式文件系统。缺点：延迟、写操作难。

romfs:只读文件系统

4 嵌入式Linux（续）

- 日志文件系统：

  日志记录，先写日志，后写数据。

  JFFS:专用于Flash的日志式文件系统。

4 嵌入式Linux（续）

YAFFS文件系统

- YAFFS，Yet Another Flash File System，是一种类似于JFFS/JFFS2的专门为Flash设计的嵌入式文件系统。与JFFS相比，JFFS和JFFS2适合NOR FLASH，YAFFS是为NAND FLASH量身定做的,它减少了一些功能，因此速度更快、占用内存更少。

4 嵌入式Linux（续）

- YAFFS和JFFS都提供了写均衡，垃圾收集等底层操作。它们的不同之处在于：
（1）JFFS是一种日志文件系统，通过日志机制保证文件系统的稳定性。YAFFS仅仅借鉴了日志系统的思想，不提供日志机能，所以稳定性不如JFFS，但是资源占用少。
（2）JFFS中使用多级链表管理需要回收的脏块，并且使用系统生成伪随机变量决定要回收的块，通过这种方法能提供较好的写均衡，在YAFFS中是从头到尾对块搜索，所以在垃圾收集上JFFS的速度慢，但是能延长NAND的寿命。
（3）JFFS支持文件压缩，适合存储容量较小的系统；YAFFS不支持压缩，更适合存储容量大的系统。

4 嵌入式Linux（续）

X Window是一个在大多数UNIX工作站上使用的图形用户界面。

它是一种与平台无关的客户机/服务器(Client/Server)模型，可以让用户在一台机器上调用另一台机器的X Window库，打开另一台机器上的窗口，而不需要考虑这两台机器自身的操作系统类型。

正是这种特性使UNIX和Linux系统上的用户和应用程序非常自然地通过网络连接在一起。使用X Window开发GUI时，因为开发环境成熟，开发工具易用，所以可以缩短开发时间，降低开发难度。

4 嵌入式Linux（续）

X Window系统应用于嵌入式系统时，要考虑嵌入式系统的特殊条件。嵌入式系统由于资源有限，不宜使用体积大的操作系统内核，这是因为需要将系统固化在ROM中或者Disk On Chip的FLASH ROM上。

但是，一个X Lib就需要大概10MB ~ 20MB的空间，在一般的嵌入式环境不能满足这样的条件。所以，针对嵌入式领域，X Window进行了必要的裁剪和优化，产生了很多嵌入式GUI系统。

4 嵌入式Linux（续）

面向嵌入式Linux 系统的图形用户界面

- MicoroWindows/NanoX
  - 开放源码
  - 无任何硬件加速能力
  - 图形引擎中存在许多低效算法
  - 代码质量较差
- OpenGUI
  - 可移植性稍差
- Qt/Embedded
  - 低的程序效率、大的资源消耗
- MiniGUI

4 嵌入式Linux（续）

嵌入式Linux-Android

- Android
  - Google+开放手机联盟
  - 2007年11月5日发布
  - “Android 是第一个完整、开放、免费的手机平台”
  - 2008年9月22日，第一款基于Android的手机发布
    - T-Mobile G1
  - 基于Android的手机GPhone
- Android的特点
  - 包括了操作系统、中间件、用户界面和应用软件
    - 这是一个平台
  - 面向智能手机

内容提要

1	嵌入式操作系统简介
2	μC/OS
3	VxWorks
4	嵌入式Linux
5	COS与安全嵌入式操作系统

安全的嵌入式软件

- ○2 功能规范
- ○3 高层设计
- ○4 低层设计
- ○5 脆弱性分析
- ○6 安全策略模型
- ○7 对应性分析
- ○8 管理员指南
- ○9 交付与运行
- ○10 开发安全
- ○11 开发者测试报告
- ○12 开发者测试分析
- ○13 配置管理
- ○14 用户指南
- ○15 生命周期定义
- ○16 开发工具与技术

操作系统安全

- TCSEC
  - D级
  - C1级：基本的访问控制
  - C2级：灵活的访问安全
  - B1：标记安全保护
  - B2：结构化保护（基于硬件，内容分区）
  - B3：安全域，分层
  - A：校验级

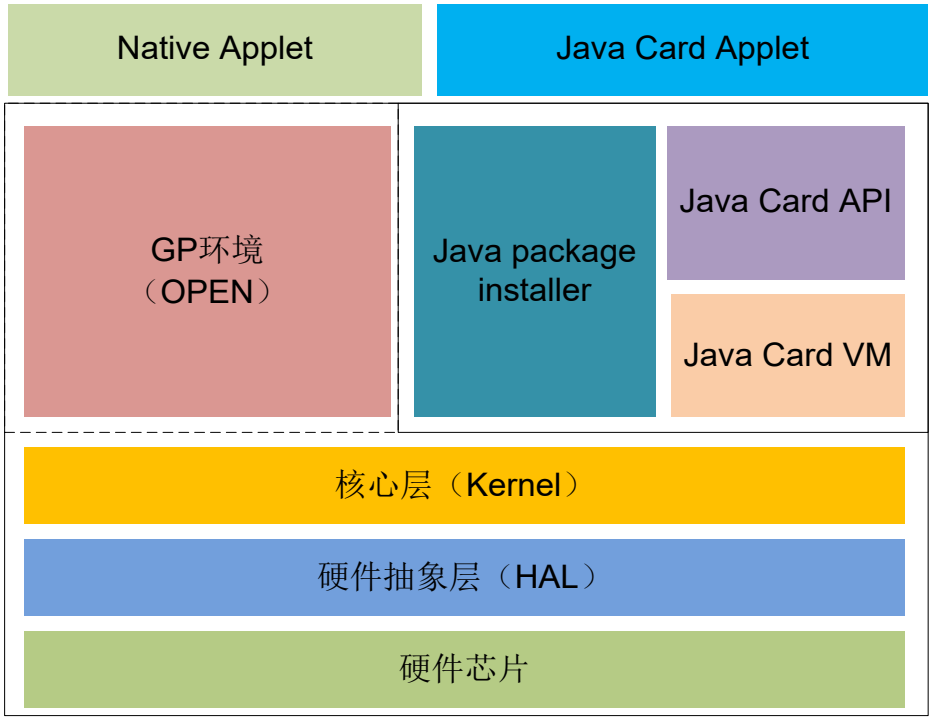
传统操作系统的安全关注点

- 帐户管理与访问控制
- 网络安全
- 病毒与防火墙
- 漏洞扫描与升级
- 流量监控
- ……

嵌入式操作系统的安全关注点

- 1 安全审计
- 2 数据空间暴力破解
- 3 密钥功能
- 4 数据传输保护
- 5 数据访问控制
- 6 环境压力
- 7 文件结构控制
- 8 错误注入
- 9 信息泄露
- 10 嵌入式软件ID
- 11 初始状态
- 12 生命周期功能
- 13 逻辑保护
- 14 多应用
- 15 物理防护
- 16 重放
- 17 安全通讯
- 18 启动序列
- 19 多次重复操作

COS的基本概念


Native Applet
Java Card Applet
GP环境
（OPEN）
Java package
installer
Java Card API
Java Card VM
核心层（Kernel）
硬件抽象层（HAL）
硬件芯片

COS的基本概念

- 文件系统
  - 基于对象
  - 基于分配表
  - 基于链表
  - 基于预分配
- 命令系统
  - 多应用
  - 单应用
  - 标准应用
- 安全系统
  - 安全状态
  - 安全属性
  - 安全机制
- 接口系统
  - 链路层
  - 传输层
  - 物理层

COS的认证

- 认证方法
  - 密码
  - 口令
  - 生物识别
  - 多因子认证
- 认证协议
  - SCP
  - 7816
- 认证存储安全
  - PUF
  - MCB
  - Memory