嵌入式操作系统

第二章 RTOS基本概念与架构

Recommend
火龙果软件

- 为什么需要RTOS
- RTOS的分类与特点
- RTOS的基本概念
- RTOS的主要架构

Recommend
火龙果软件

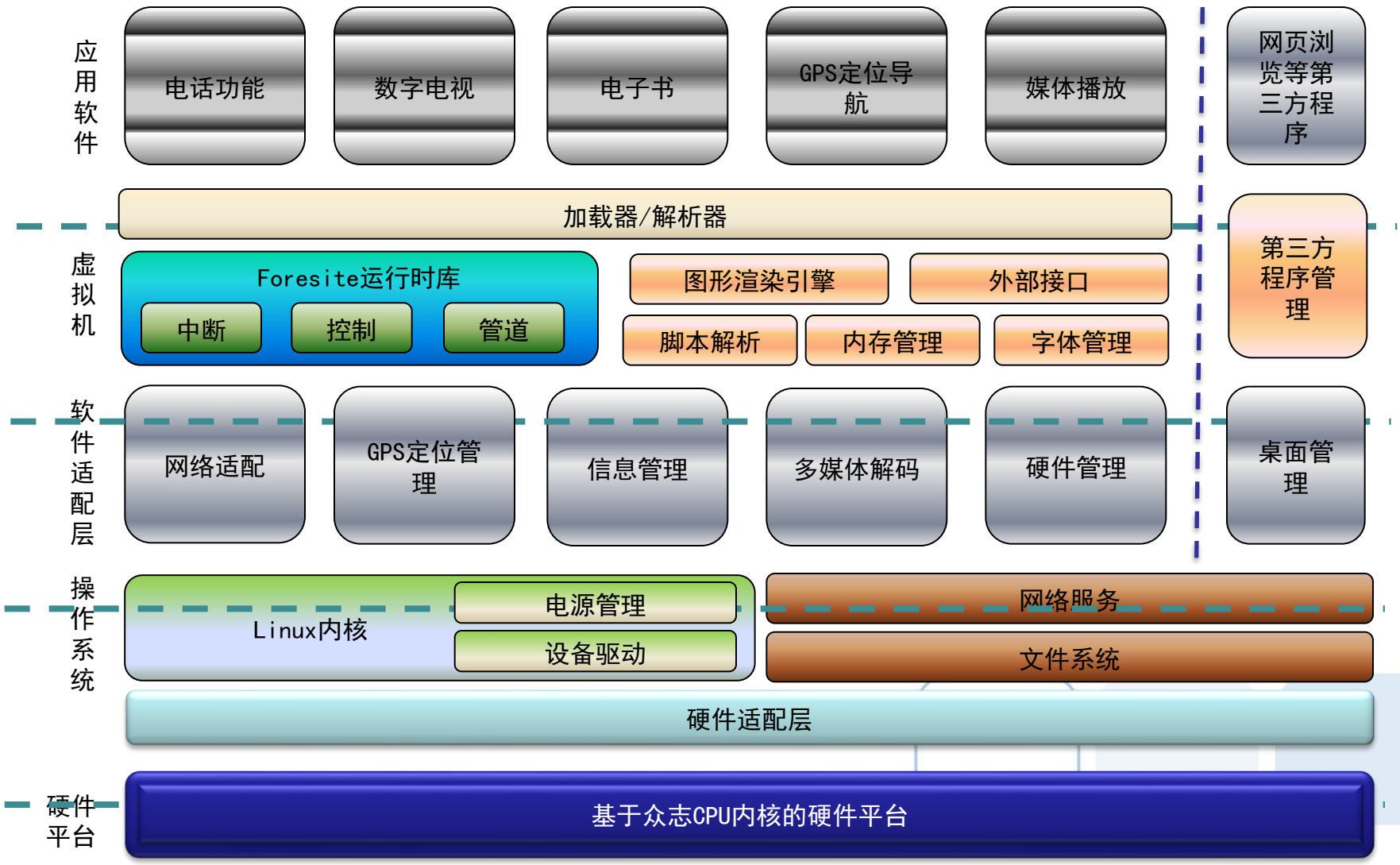
使用RTOS的必要性

- 可将应用程序划分为多任务，简化了应用程序的开发与设计，大大降低了耦合性
- 提高了系统的稳定性与可靠性
- 由RTOS来管理并分配系统资源，可以兼顾效率与公平
- 使系统更易于扩展与维护
- 为实时性提供保证
- 弥补嵌入式系统工程师与计算机专家之间的“Gap”，将“丑陋”的计算机硬件装扮为美丽的“接口”

Recommend
火龙果软件

使用RTOS的可行性

- 硬件性能与功能不断提升 → MCU，SoC
- RTOS的种类越来越多 → uC/OS，Linux，WinCE，VxWorks，Nucleus...
- RTOS与程序员的接口越来越友好 → C、C++、Java、C#...服务越来越多...
- 便宜了，不差钱！ → 天价使用费→Free！



Recommend
火龙果软件

通用操作系统

- 通常用于PC机、服务器等大中小型计算机
- 操作系统功能的全集，极其复杂
- 实时性不强
- Windows NT/XP、Linux、Unix、Mac、Solaris

- 用过Windows的都知道，常常“死”的很难看，连盖茨也不放过！
- 想象一下这样的系统用在航空器或医疗设备上。。。

嵌入式操作系统

- 通常用于嵌入式系统
- 通用操作系统功能的子集，功能复杂度降低
- 实时性较强
- 嵌入式Linux，Windows CE/Mobile，uC/OS，VxWorks等等

Recommend
火龙果软件

嵌入式操作系统一览

VxWorks®
WIND RIVER

Microsoft Windows Embedded

Microsoft Windows CE

Microsoft Windows xp Embedded

Windows Mobile

THREADX® BY EXPRESS LOGIC

µC/OS-II The Real-Time Kernel

ecos

real Linux 2.0

PEEWEE LINUX THE EMBEDDED LINUX

LynxOS FROM LYNUXWORKS

palm

Psos

Recommend
火龙果软件

嵌入式操作系统根据实时性的划分

- 嵌入式系统一般具有实时的特点
- 所谓“实时系统”一般是指，在此类系统中：
  - 一个高优先级的任务可以得到立刻的、没有延迟的服务，不需要等待其他低优先级的任务，一旦得到处理器的使用权就将执行到结束或出现就绪的更高优先级任务
  - 因此系统必须具有高效处理异步事件（中断）的能力和高速的I/O能力

Recommend
火龙果软件

嵌入式操作系统根据实时性的划分

- 硬实时系统
  - 确保系统中的关键任务在确定的时间得到响应，绝对不能例外，否则将造成严重后果
- 软实时系统
  - 有响应时间的要求，但即使偶尔例外也不会引起严重后果
- 非实时系统
  - 无响应时间要求

Recommend
火龙果软件

举例

- 硬实时系统

  VxWorks，uC/OS-II，RTLinux，嵌入式实时Linux

- 软实时系统

  Windows Embedded系列，Palm OS，一些嵌入式Linux，Android

- 非实时系统

  几乎没有

Recommend
火龙果软件

RTOS的性能指标

- 实时性
- 稳定性
- 尺寸
- 可扩展性

Recommend
火龙果软件

临界区

- 指运行时不可被打断的程序区域

一旦这部分程序开始执行，则不允许以任何方式中断其运行。为确保临界区代码的独占执行，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，以免系统实时性下降。

Recommend
火龙果软件

资源

- 任何为任务所占用的实体都可称为资源

可以是输入输出设备（如打印机、键盘、显示器），可以是文件，也可以是一个变量、一个结构或一个数组，甚至可以是一个函数、一段程序

Recommend
火龙果软件

共享资源

- 可以被一个以上任务使用的资源叫做共享资源

为了防止数据被破坏而产生不一致，每个任务在与共享资源打交道时，必须独占该资源，这叫做互斥（*mutual exclusion*）

Recommend
火龙果软件

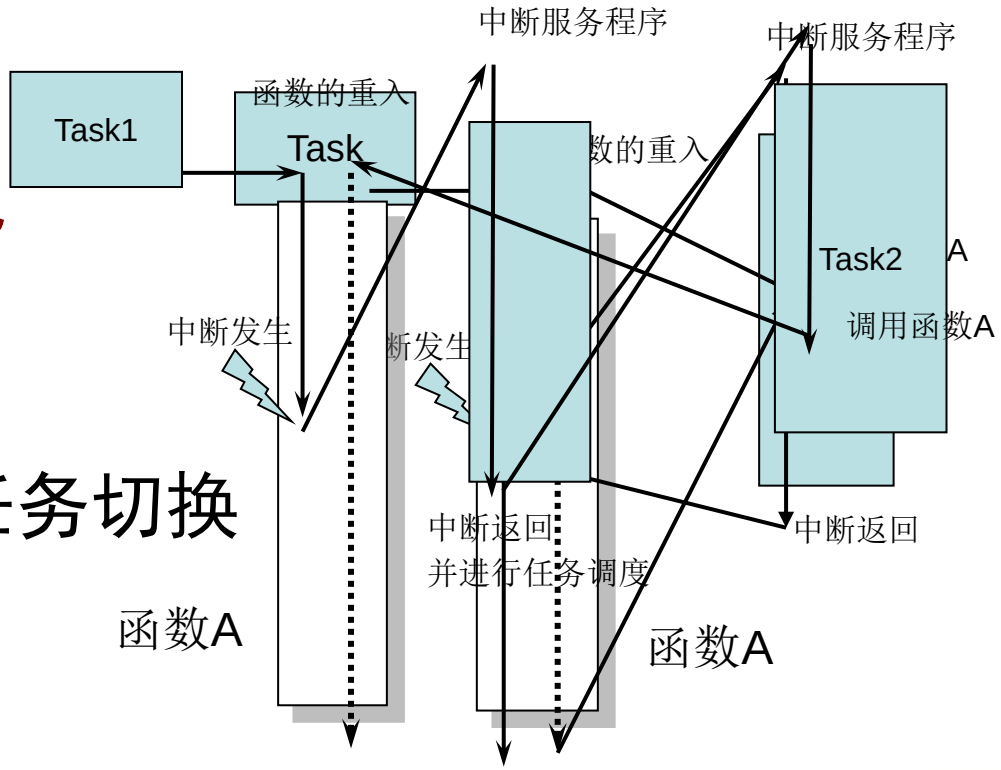
可重入函数

- 可重入型函数可以被一个以上的任务调用，而不必担心数据被破坏
  - 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，而相应数据不会被破坏
  - 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

Recommend
火龙果软件

可重入函数

- 函数重入的三种情况
  - 中断服务程序
  - 可剥夺型内核中任务切换
  - 递归调用



Recommend
火龙果软件

可重入函数

- 可重入函数的例子

```
void strcpy(char *dest, char *src)
{
    while (*dest++ = *src++)
     {        ;    }
    *dest = NUL;
}
```

Recommend
火龙果软件

可重入函数

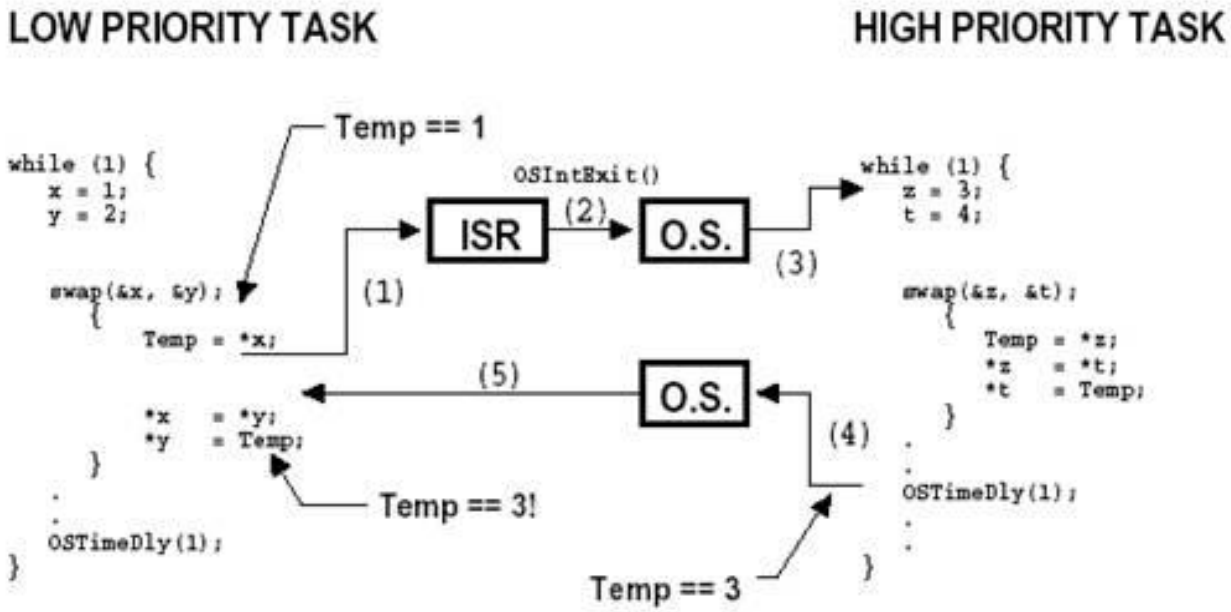
- 不可重入函数的例子

```
int Temp;
void swap(int *x, int *y)
{
    Temp = *x;
    *x   = *y;
    *y   = Temp;
}
```

Recommend
火龙果软件

可重入函数

- 不可重入函数说明



Recommend
火龙果软件

可重入函数

- 不可重入函数的对策

使用以下方法可使函数变为可重入函数

– 把Temp定义为局部变量
– 调用Swap()函数之前关中断，调动后再开中断
– 用信号量禁止该函数在使用过程中被再次调用

Recommend
火龙果软件

多任务

- 多任务运行的实现实际上是靠CPU(MPU)在多个任务之间转换、调度，交错运行
  - CPU只有一个，轮番服务于一系列任务中的某一个，一个CPU被虚拟化为若干个CPU
  - 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
  - 在实时应用中，多任务化使开发人员可以将很复杂的应用程序层次化

物理世界本身是并发的！

Recommend
火龙果软件

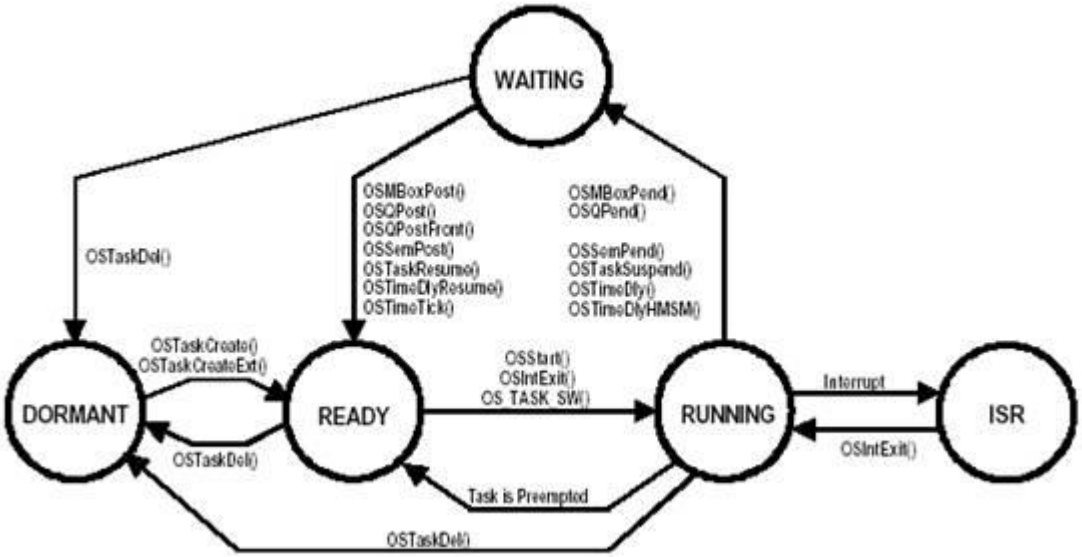
任务

- 也可称为线程，是一个单执行体的程序，该程序可以认为CPU完全只属该程序自己
  - 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，有它自己的一套CPU寄存器和自己的栈空间

Recommend
火龙果软件

任务的状态

- 典型地、每个任务都是一个无限的循环。每个任务都处在以下5种状态之一的状态下，这5种状态是休眠态，就绪态、运行态、挂起态(等待某一事件发生)和被中断态

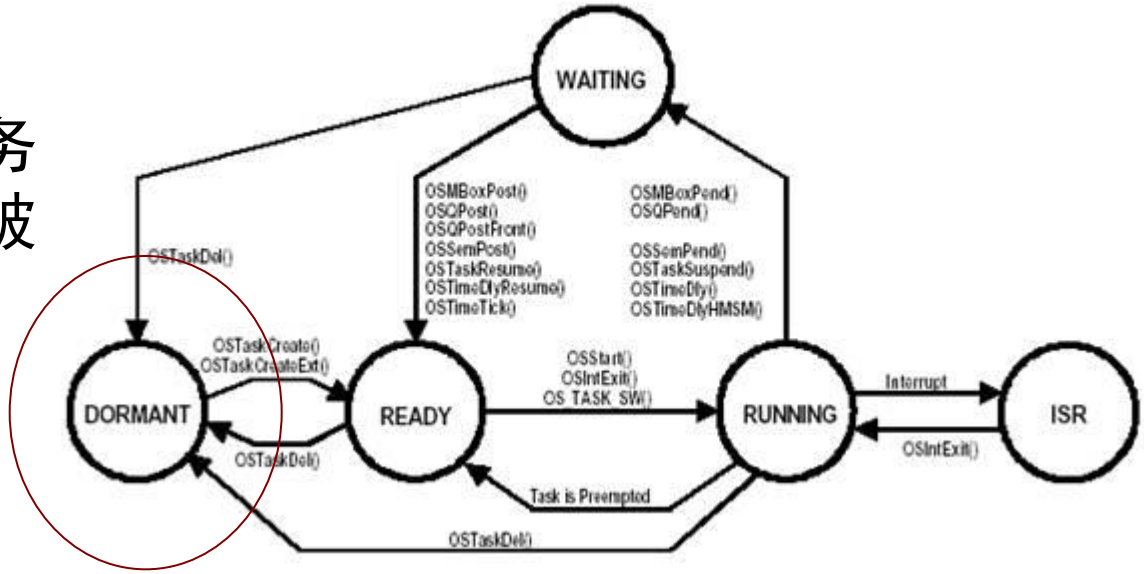


Recommend
火龙果软件

任务的状态

休眠态

休眠态相当于该任务驻留在内存中，但并不被多任务内核所调度。

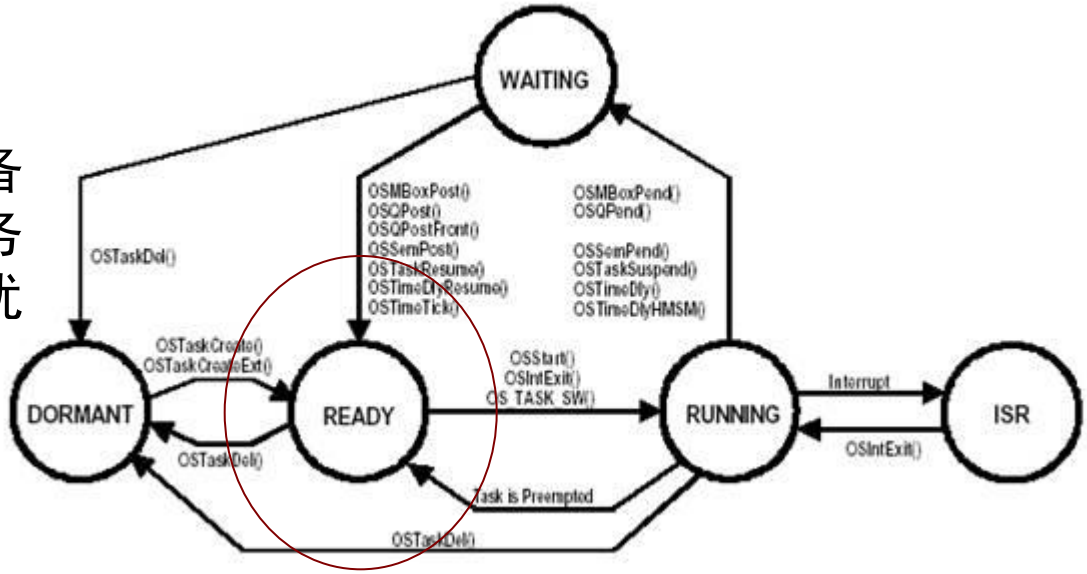


Recommend
火龙果软件

任务的状态

就绪态

就绪意味着该任务已经准备好，可以运行了，但由于该任务的优先级比正在运行的任务的优先级低，还暂时不能运行

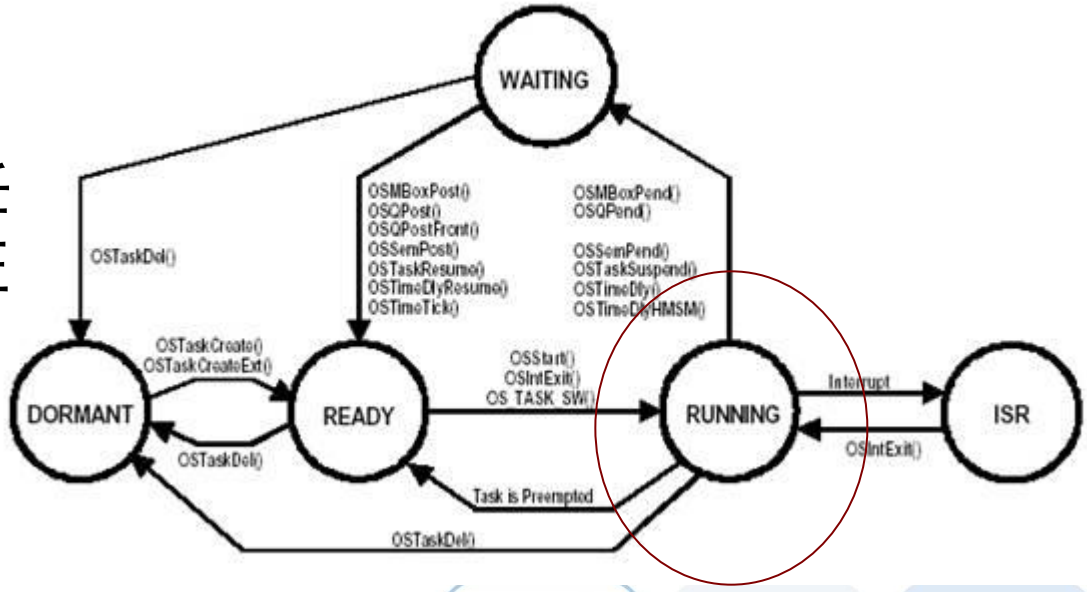


Recommend
火龙果软件

任务的状态

运行态

运行态的任务是指该任务掌握了CPU的控制权，正在运行中



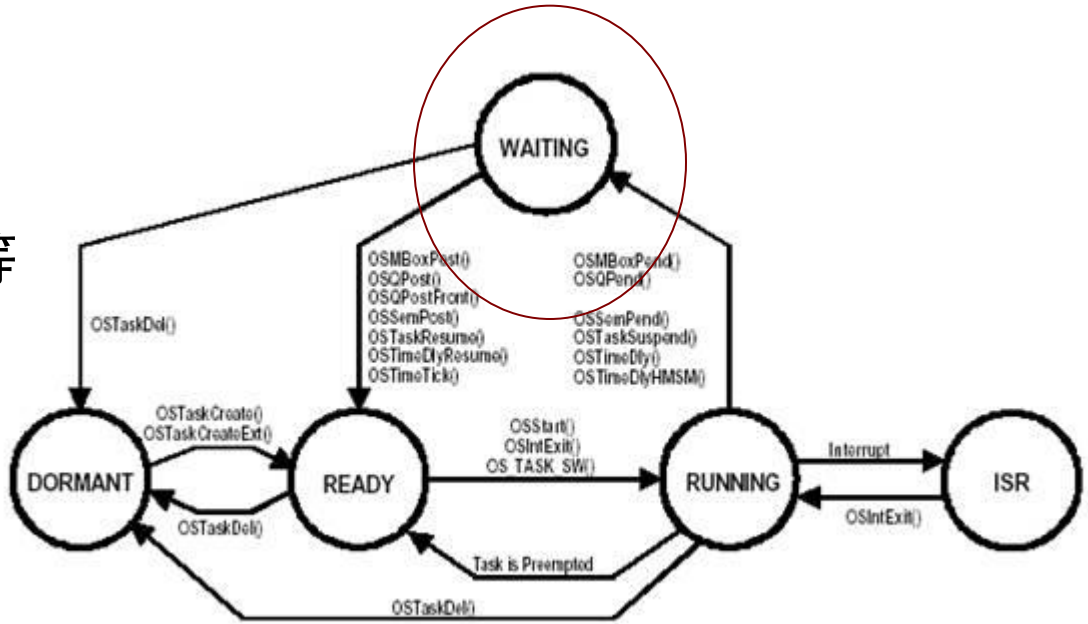
Recommend
火龙果软件

任务的状态

挂起态（阻塞态）

■ 挂起状态也叫做等待事件态WAITING，指该任务在等待，等待某一事件的发生

■ 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，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

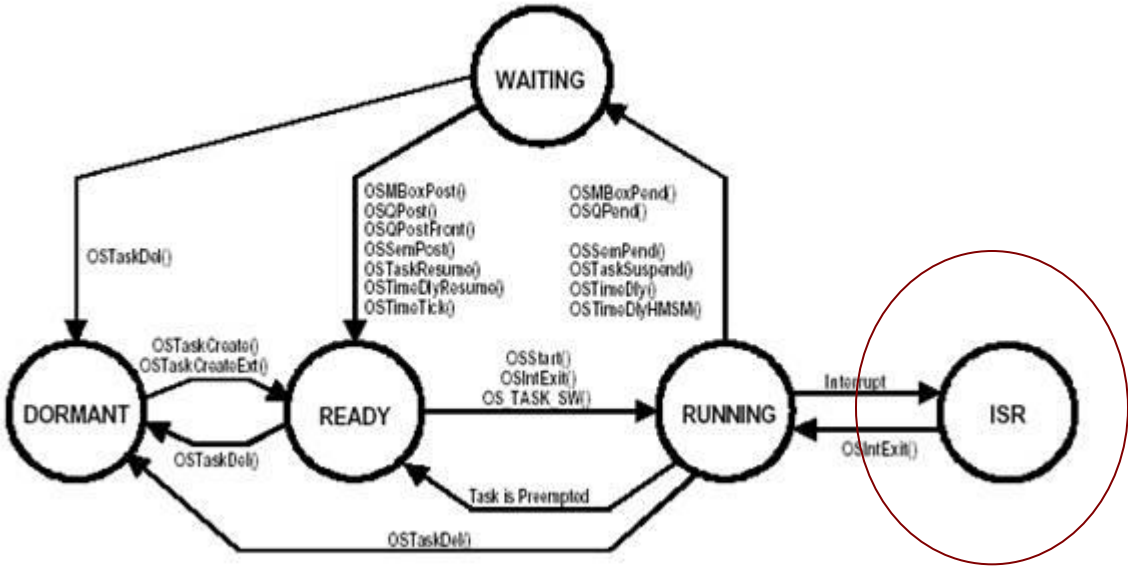


Recommend
火龙果软件

任务的状态

中断态

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



Recommend
火龙果软件

任务切换

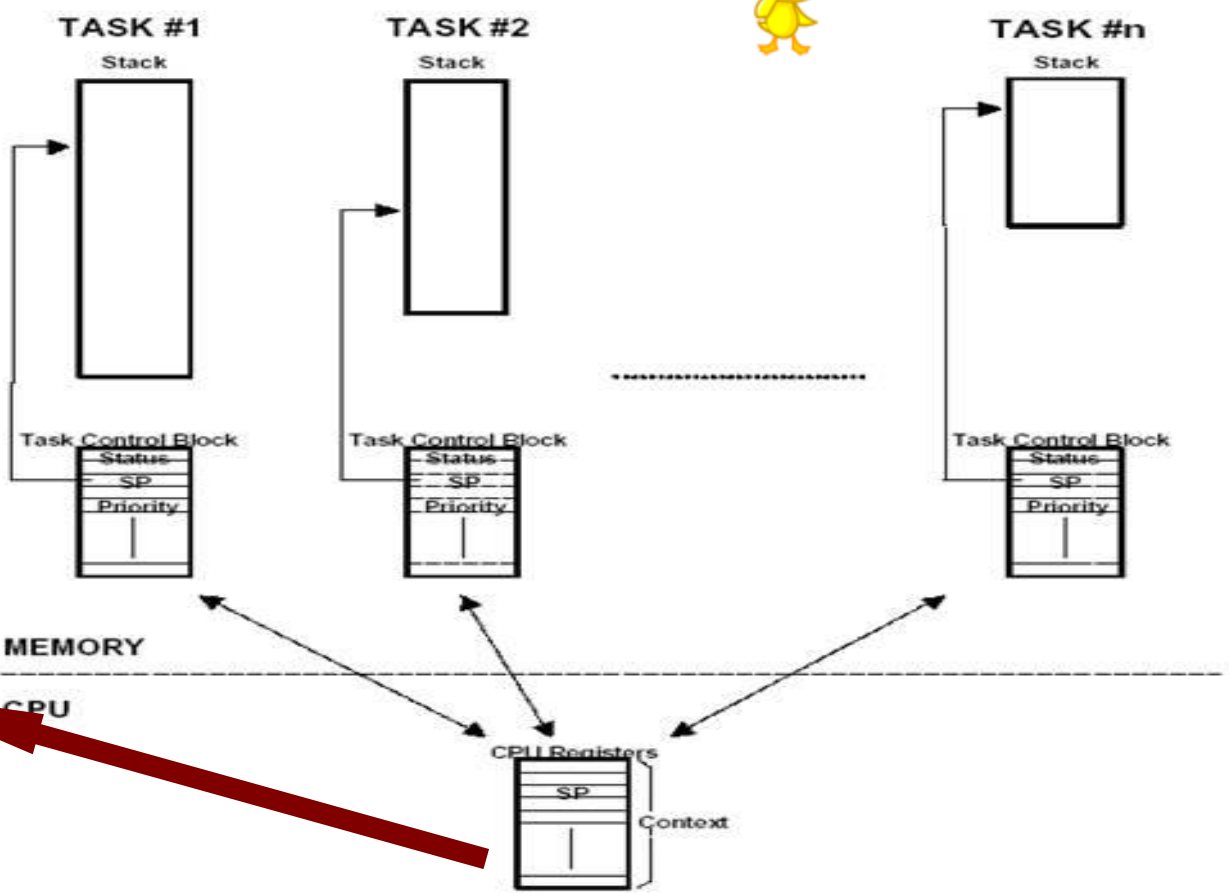
- Context Switch 在有的书中翻译成上下文切换，实际含义是任务切换，或CPU寄存器内容切换
  - 当多任务内核决定运行另外的任务时，它保存正在运行任务的当前状态（Context），即CPU寄存器中的全部内容
  - 这些内容保存在任务自己的栈区之中
  - 入栈工作完成以后，就把下一个将要运行的任务的当前状况从该任务的栈中重新装入CPU的寄存器，并开始下一个任务的运行

Recommend
火龙果软件

任务切换

程序呢？

SP
R1
RN
PC
PSW



Recommend
火龙果软件

任务切换的开销

- 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

Recommend
火龙果软件

内核（Kernel）

- 多任务系统中，内核负责[illegible]说为每个任务分配CPU时间，负责任务[illegible]中断
  - 内核提供的基本服务是任务切换
  - 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
  - 每个任务要有自己的栈空间，比较耗费内存
  - 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，诸如信号量管理，邮箱、消息队列、延时等，实时内核使得CPU的利用更为有效
- 一旦读者用实时内核做过系统设计，将决不再想返回到前后台系统

Recommend
火龙果软件

调度

- 内核的主要职责之一，就是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
  - 然而，究竟何时让高优先级任务掌握CPU的使用权，有两种不同的情况，分别为不可剥夺型内核与可剥夺型内核

Recommend
火龙果软件

不可剥夺型内核

- 不可剥夺型内核要求每个任务自我放弃CPU的所有权
  - 直到运行的任务主动放弃CPU的使用权时，高优先级的任务才能获得CPU的使用权，除非有中断到来
  - 任务级响应时间取决于最长的任务执行时间

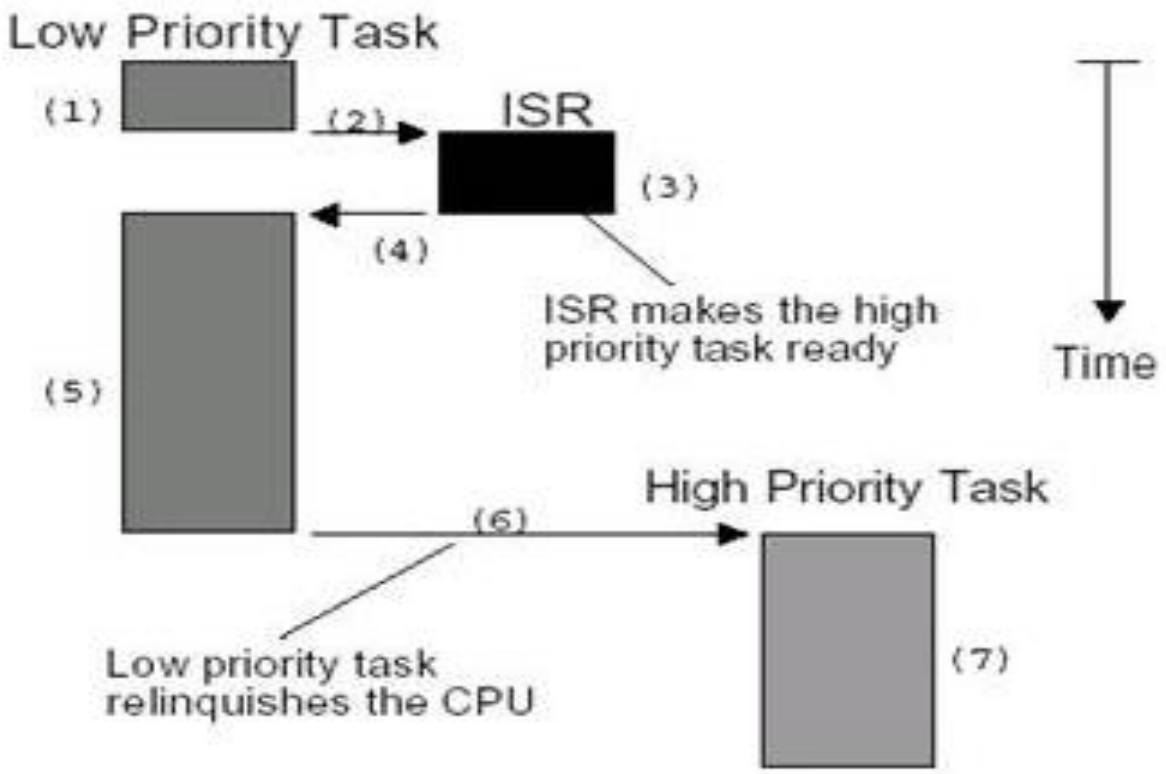
致命的！

- 因此商业RTOS内核几乎没有不可剥夺型的

优点	缺点
允许使用不可重入函数	响应时间不确定
几乎不需要使用信号量保护共享数据	

Recommend
火龙果软件

不可剥夺型内核任务调度



Recommend
火龙果软件

可剥夺型内核

可使用互斥量解决！

- 最高优先级的任务一旦就绪，总能得到CPU的控制权
  - 当一个运行的任务使一个比它优先级高的任务进入了就绪态，当前任务的CPU使用权就被剥夺了，那个高优先级的任务立即得到了CPU的控制权
  - 如果是中断服务子程序使一个高优先级的任务进入就绪态，中断完成时，中断了的任务被挂起，优先级高的那个任务开始运行

优点	缺点
响应时间更强	不允许使用重入函数

uC/OS-II是可剥夺型的！

Recommend
火龙果软件

时间片轮番调度法

- 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

  内核满足以下条件时，把CPU控制权交给下一个就绪任务：

  – 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了

  – 当前任务的时间片已用完

Recommend
火龙果软件

互斥条件

- 保证多个任务排它访问共享资源的条件
  - 实现任务间通讯最简便的办法是使用共享数据结构，特别是当所有任务在一个地址空间下，能使用全局变量、指针、缓冲区、链表、循环缓冲区等
  - 但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，即为互斥
  - 满足互斥条件最一般的方法有：
    - 关中断
    - 使用测试并置位指令
    - 禁止做任务切换
    - 利用信号量

Recommend
火龙果软件

源头遏制

手段粗暴

满足互斥条件的方法

- 关中断
  - 处理共享数据时保证互斥，最简便快捷的办法是关中断和开中断，如下所示：

```
Disable interrupts;                    /*关中断*/
  Access the resource (read/write variables);    /*读/写变*/
Reenable interrupts;                   /*重新允许中断*/
```

  - 需要注意的是，关中断的时间不能太长，因为它影响整个系统的中断响应时间，即中断延迟时间

Recommend
火龙果软件

受到硬件条件限制！

满足互斥条件的方法

- 测试并置位
  - 当两个任务共享一个资源时，约定如下：先测试某一全局变量，如果该变量是0，允许该任务与共享资源打交道。为防止另一任务也要使用该资源，前者只要简单地将全程变量置为1，这通常称作测试并置位(Test-And-Set),或称作TAS
  - TAS操作可能是微处理器的一条不会被中断的指令，或者是在程序中关中断做TAS操作再开中断

Recommend
火龙果软件

满足互斥条件的方法

不推荐采用这样的方法——因噎废食！

- 禁止任务切换
  - 如果任务不与中断服务子程序共享变量或数据结构，可以使用禁止、然后允许任务切换进行互斥，如下所示：

```
void Function (void)
{
    OSSchedLock();
    /*在这里处理共享数据(中断是开着的)*/
    OSSchedUnlock();
}
```

Recommend
火龙果软件

什么都不让用，
RTOS真麻烦，
不玩了！

Recommend
火龙果软件

满足互斥条件的方法

这才是正道！

- 信号量（Semaphores）
  - 一种用于约定的机制：某信号量如果被一个任务占用，当另一个任务去获取该信号量时就会挂起，直到占用该信号量的任务释放该信号量
  - 因此信号量的作用为：
    - 控制共享资源的使用权(满足互斥条件)
    - 标志某事件的发生
    - 使两个任务的行为同步

Recommend
火龙果软件

满足互斥条件的方法

- 信号量（Semaphores）
  - 信号像是一把钥匙，任务在竞争共享资源时，要得到共享资源并运行下去，得先拿到这把钥匙
  - 信号量是可计数的，只有0、1两个值的信号量又可称为信号，是最常用的
  - 一般对信号量只能实施三种操作：
    - 初始化(INITIALIZE)，也可称作建立(CREATE)
    - 等信号(WAIT)，也可称作挂起(PEND)
    - 给信号(SIGNAL)，也可称为发信号(POST)

Recommend
火龙果软件

满足互斥条件的方法

- 信号量（Semaphores）
  - 利用信号量处理共享数据的示例程序

```
OS_EVENT *SharedDataSem;
void Function (void)
{
    INT8U err;
    OSSemPend(SharedDataSem, 0, &err);
    /*共享数据的处理在此进行，(中断是开着的)*/
    OSSemPost(SharedDataSem);
}
```

等信号

给信号

Recommend
火龙果软件

满足互斥条件的方法

- 信号量（Semaphores）
  - 使用实例（打印机）

    当诸任务共享输入输出设备时，信号量特别有用。可以想象，如果允许两个任务同时给打印机送数据时会出现什么现象。

    打印机会打出相互交叉的两个任务的数据。例如任务1要打印“I am Task!”，而任务2要打印“I am Task2!”可能打印出来的结果是：“I Ia amm T Tasask k1!2!”

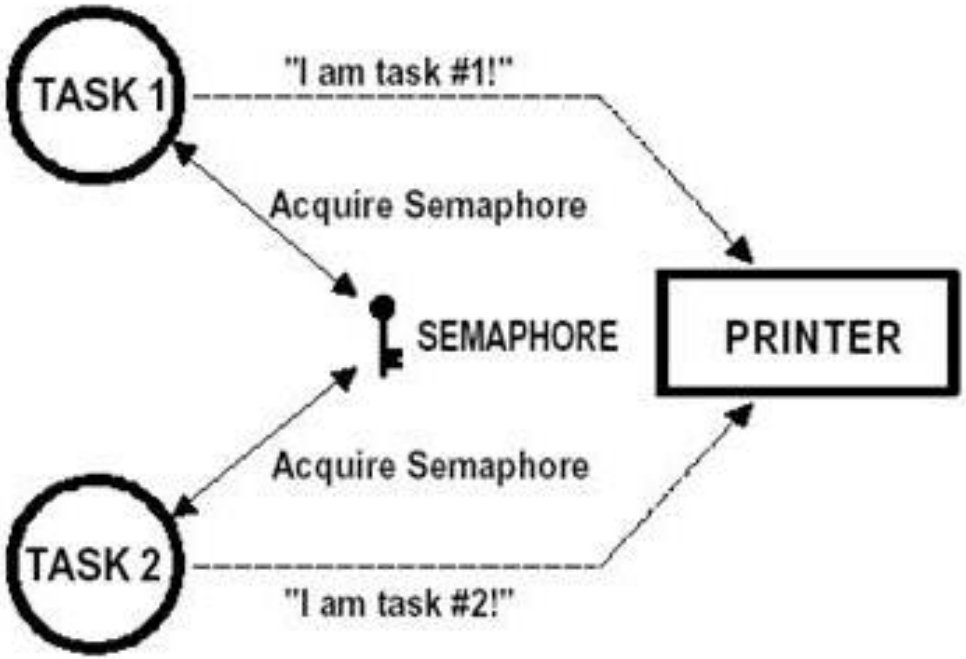
Recommend
火龙果软件

满足互斥条件的方法

- 信号量（Semaphores）
  - 使用实例（打印机）

    在这种情况下，使用信号量并给信号量赋初值1(用二进制信号量)。

    规则很简单，要想使用打印机的任务，先要得到该资源的信号量。



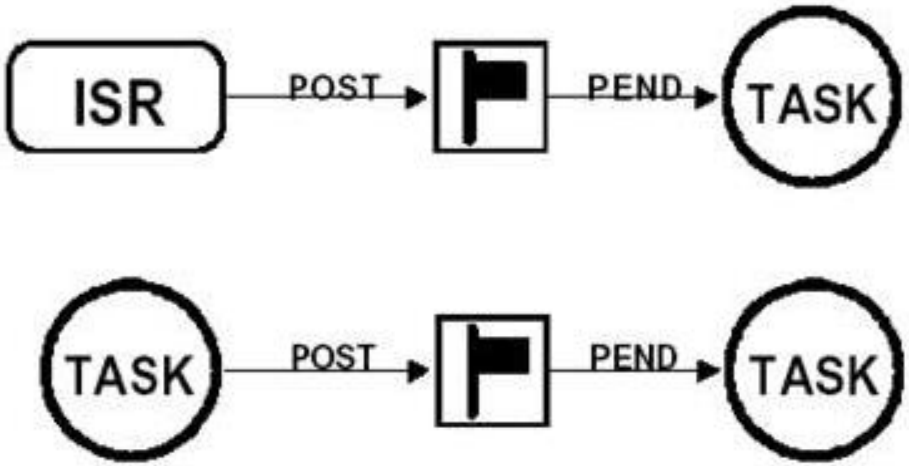
Recommend
火龙果软件

同步（信号量的重要应用）

- 可以利用信号量使某任务与中断服务或其他任务同步

右图说明了利用信号量实现同步。用一面旗帜，或称作一个标志表示信号量。这个标志表示某一事件的发生(不再是一把用来保证互斥条件的钥匙)。

用来实现同步机制的信号量初始化成0，信号量用于这种类型同步的称作单向同步(unilateral rendezvous)。一个任务做I/O操作，然后等信号回应。当I/O操作完成，中断服务程序(或另外一个任务)发出信号，该任务得到信号后继续往下执行。

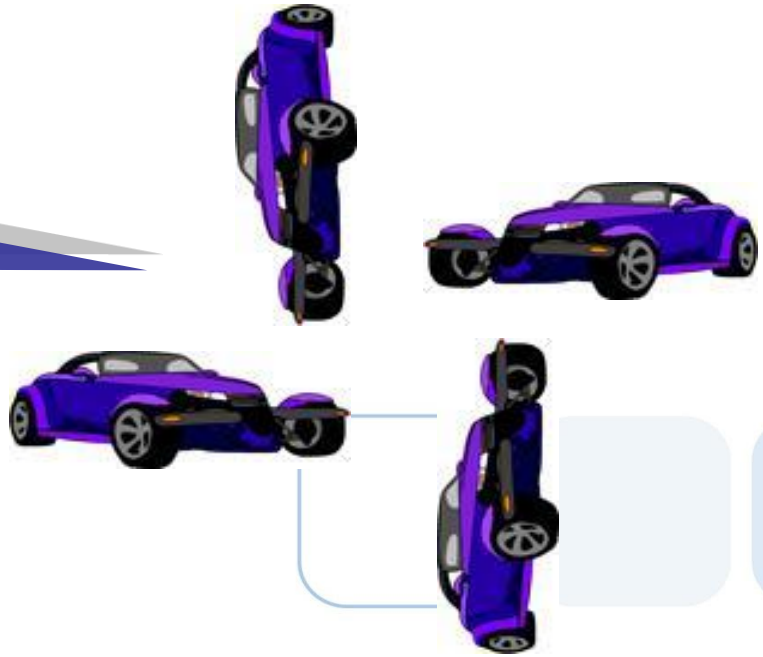


Recommend
火龙果软件

死锁

- 是指两个或两个以上的任务在执行过程中,因争夺资源而造成的一种互相等待的现象,若无外力作用,它们都将无法推进下去

四辆在路上斗气的汽车，
形成了一个死锁系统！



Recommend
火龙果软件

死锁

- 防止发生死锁的简单方法
  - 先得到全部需要的资源再做下一步的工作
  - 用同样的顺序去申请多个资源
  - 释放资源时使用相反的顺序
  - OS的对策：

    内核大多允许用户在申请信号量时定义等待超时，以此化解死锁。当等待时间超过了某一确定值，信号量还是无效状态，就会返回某种形式的出现超时错误的代码，这个出错代码告知该任务，不是得到了资源使用权，而是系统错误。

Recommend
火龙果软件

任务间通讯

- 有时很需要任务间的或中断服务与任务间的通讯，这种信息传递称为任务间的通讯
  - 任务间信息的传递一般有两个途径：
    - 通过全程变量
    - 发消息给另一个任务
      - 消息邮箱（可以理解为带有存放数据指针的信号量）
      - 消息队列 （可以理解为消息邮箱的队列）

将在任务间通讯章专题讲解

Recommend
火龙果软件

中断

- 中断是一种硬件机制，用于通知CPU有个异步事件发生了
  - 中断一旦被识别，CPU保存部分(或全部)现场(Context)即部分或全部寄存器的值，跳转到专门的子程序，称为中断服务子程序(ISR)。中断服务子程序做事件处理，处理完成后，程序回到：
    - 在前后台系统中，程序回到后台程序
    - 对不可剥夺型内核而言，程序回到被中断了的任务
    - 对可剥夺型内核而言，让进入就绪态的优先级最高的任务开始运行

Recommend
火龙果软件

中断

- 中断的分类
  - 硬件中断 (Hardware Interrupt):一般是由外部（相对CPU内核而言）的硬件引起的事件，比如串口来数据，键盘击键等；
  - 软件中断（Soft Interrupt）：通过在程序中执行的中断指令引起的中断，又叫软陷；
    - 80X86：int 指令
    - 68000：trap 指令
    - ARM： SWI 指令

    软中断指令一般用于操作系统的系统调用入口
  - 异常（Exception）：由于CPU内部在运行过程中引起的事件，比如指令预取错，数据中止，未定义指令等，异常事件一般由操作系统接管

Recommend
火龙果软件

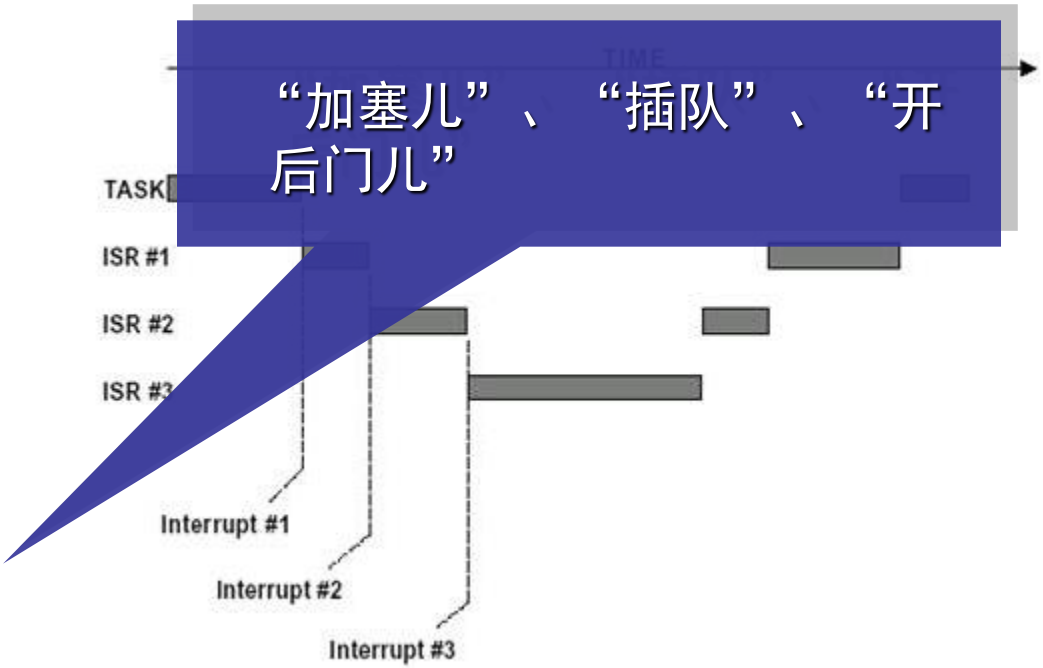
中断

- 中断延迟
  - 实时内核最重要的指标之一就是中断关了多长时间。实时系统在进入临界区代码段之前需要关中断，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
  - 在实时环境中，中断延迟应尽量的短。中断延迟太长可能会引起中断丢失

Recommend
火龙果软件

中断

- 中断嵌套
  - 是指正在执行一个中断服务时，有另一个优先级更高的中断提出中断请求，这时会暂时终止当前正在执行的级别较低的中断源的服务程序，去处理级别更高的中断源，待处理完毕，再返回到被中断了的中断服务程序继续执行，这个过程就是中断嵌套



Recommend
火龙果软件

中断响应

- 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
  - 在前后台系统及不可剥夺型内核中：
    - 中断响应时间 = 中断延迟 + 保存CPU内部寄存器的时间
  - 在可剥夺型内核中：
    - 中断响应 ＝ 中断延迟 + 保存CPU内部寄存器的时间 + 内核的进入中断服务函数的执行时间

Recommend
火龙果软件

中断恢复

- CPU返回到被中断了的程序代码所需要的时间
  - 在前后台系统及不可剥夺型内核中：
    - 中断恢复时间 = 恢复CPU内部寄存器值的时间 + 执行中断返回指令的时间
  - 在可剥夺型内核中：
    - 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

Recommend
火龙果软件

在时间片轮转调度的RTOS中，系统时钟节拍还用于生成时间片

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。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
  - 各种实时内核都有将任务延时若干个时钟节拍的功能。然而这并不意味着延时的精度是1个时钟节拍，只是在每个时钟节拍中断到来时对任务延时做一次裁决而已

Recommend
火龙果软件

时钟节拍

- 使用时钟节拍延时的示例

```
void Function (void)
{
    /*进行处理*/
    OSTimeDly(n);   /*延时n个时钟节拍，这期间任务处
                      于挂起状态*/
    /*进行后续处理*/
}
```

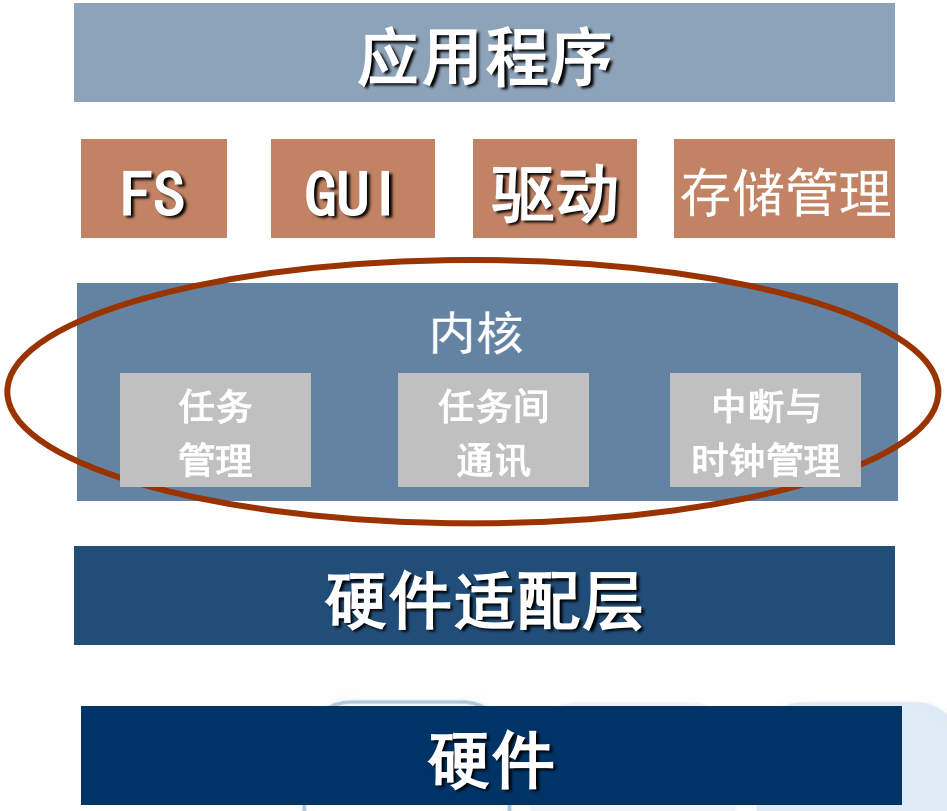
Recommend
火龙果软件

- 微内核
- 宏内核
- uC/OS-II的架构

Recommend
火龙果软件

微内核

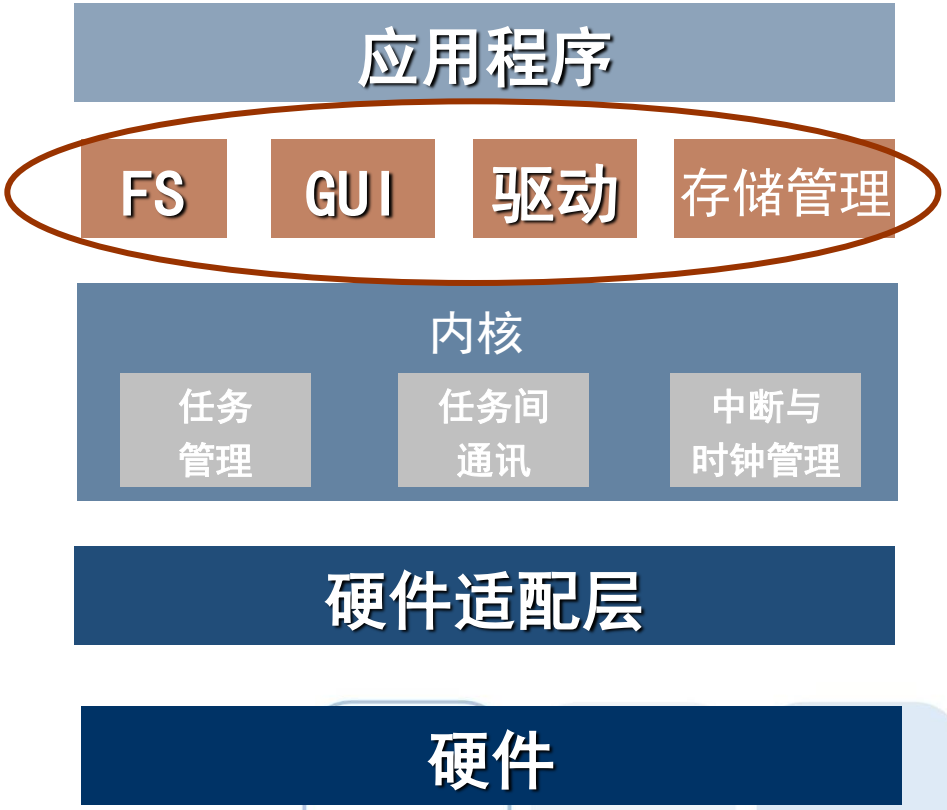
- 实现操作系统最基本的功能，一般来说包括：
  - 任务管理
  - 任务间通讯
  - 中断与时钟管理



Recommend
火龙果软件

微内核

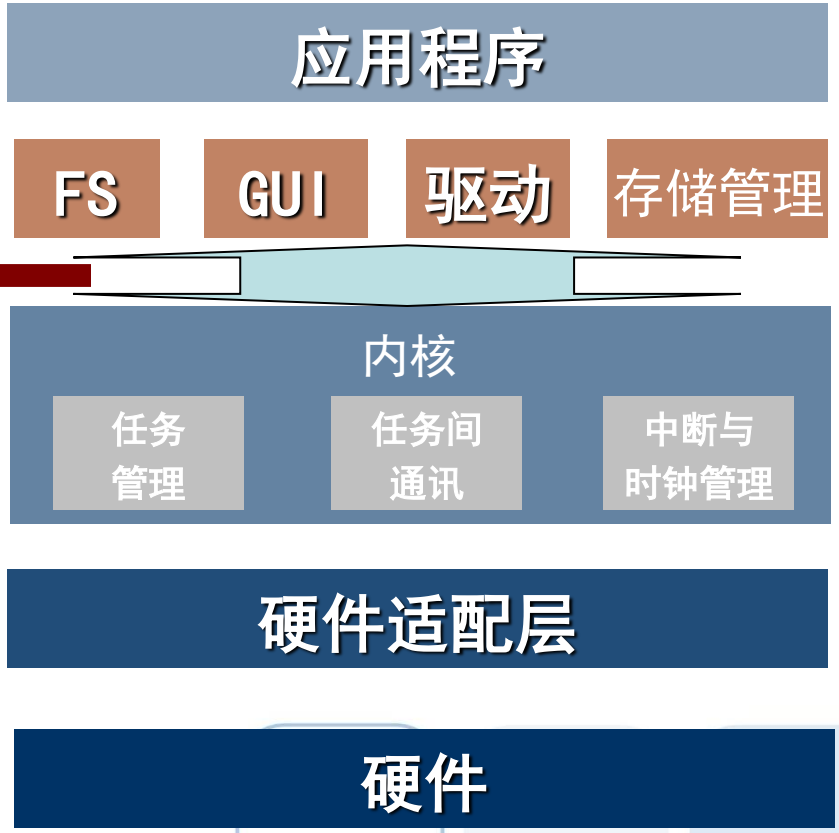
- 其他的功能在内核之外实现：
  - 文件系统
  - 图形用户接口
  - 驱动
  - 存储管理
  - ……



Recommend
火龙果软件

微内核

- 内核之外其他功能以进程或任务的形式存在，通过进程（任务）间通讯或系统调用的方式与内核交互



Recommend
火龙果软件

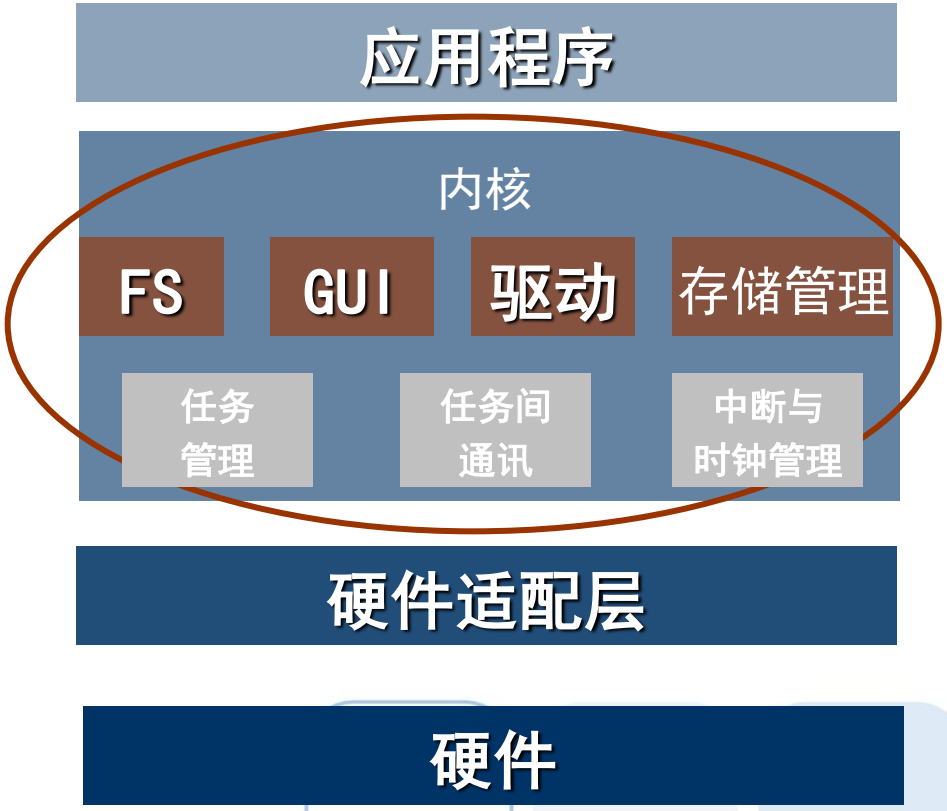
微内核

- 微内核的优点
  - 精简，便于移植
  - 稳定性强，功能之间的耦合度低
  - 资源开销小
  - 降低了设计难度，易于维护
- 微内核的缺点
  - 效率低

Recommend
火龙果软件

宏内核

- 除了实现操作系统最基本的功能外，还实现了其他管理功能，如：
  - 文件系统
  - 设备驱动
  - 存储管理
  - 其他系统服务



Recommend
火龙果软件

总的来说，微内核的优缺点与宏内核正好相反，RTOS的设计需要兼顾两者的优缺点！

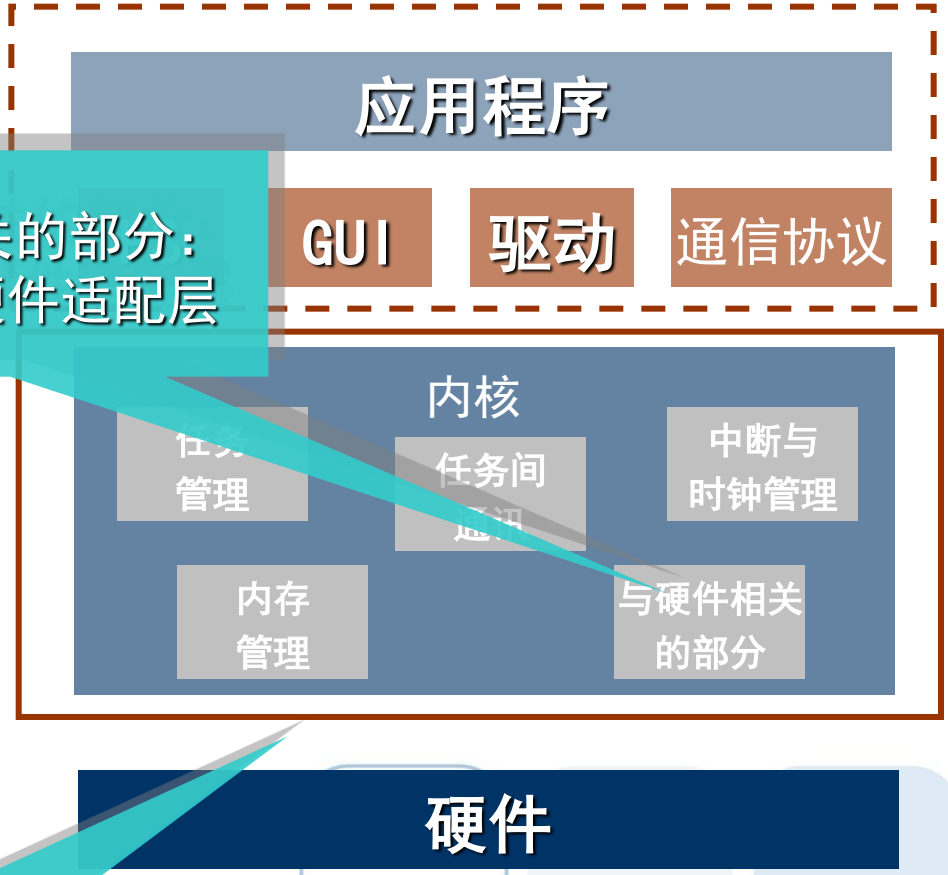
宏内核

- 宏内核的优点
  - 效率高，所有功能都在内核地址空间中实现
- 宏内核的缺点
  - 尺寸大，不利于移植
  - 稳定性受到影响，功能之间的耦合度高
  - 资源开销大
  - 设计复杂度高，增加了开发与维护难度

Recommend
火龙果软件

uC/OS-II的架构

- 内核
  - 任务管理
  - 任务间通讯
  - 中断与时钟管理
  - 内存管理
  - 与硬件相关的部分
- 内核之上（称为应用层）
  - 由开发者自行构建



与硬件相关的部分：
可以看作硬件适配层

动态内存（堆）的管理，
对C语言中malloc与free
的实现，并非内核必需的
基本功能

Recommend
火龙果软件

讨论：

uC/OS-II：
微内核
OR
宏内核？

Recommend
火龙果软件

谢　谢！

Recommend
火龙果软件