嵌入式系统
（Embedded System）

信息科学与工程学院光电子系

嵌入式操作系统初步（RT-Thread）

主要内容：

（1）RT-Thread 概述

（2）线程调度与管理

（3）任务间通信

1 RT-Thread概述 实时线程

- 版权

  属于上海睿赛德电子公司，**2006**年首发，装机量达数亿台，占据国产**RTOS**的鳌头。

- 收费问题

  免费开源，内核源代码公开。

- 意义

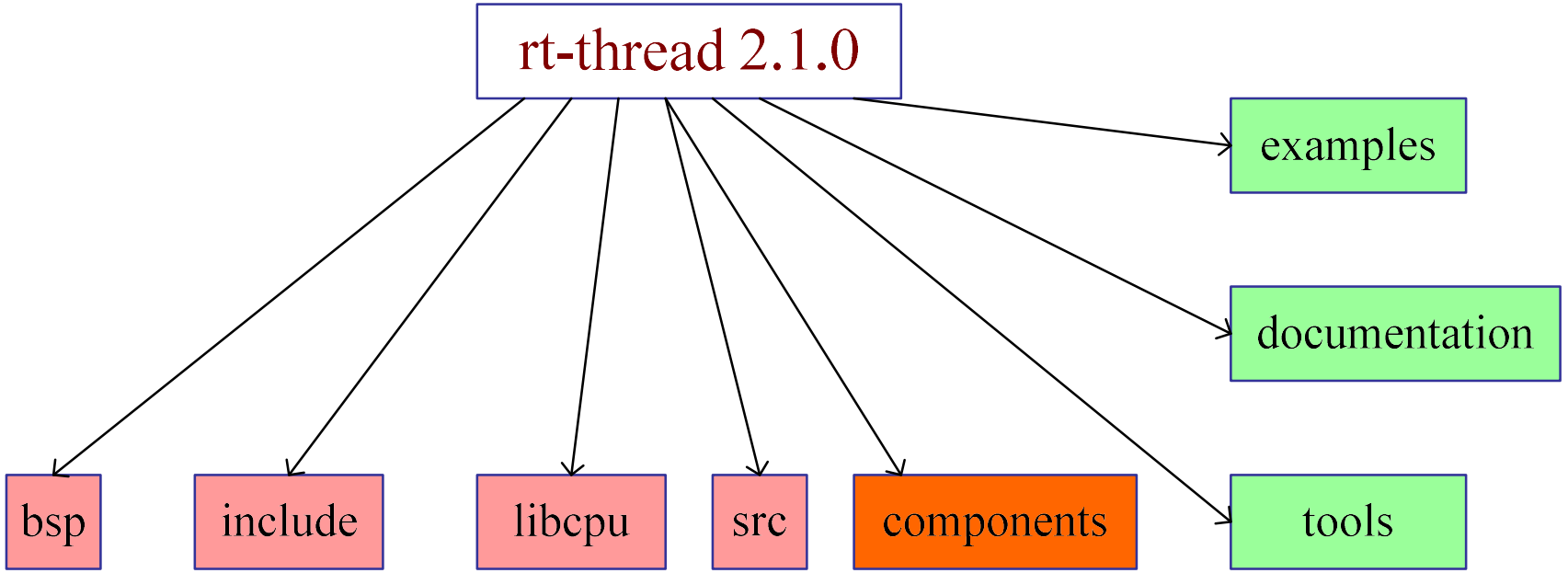
  自主品牌，经过实践检验。

参考资料

- 《RT-Thread编程手册》(电子版)
  RT-Thread小组编写

- 《RT-Thread 内核实现与应用开发实战指南》(电子版、纸版另购)
  刘火良 编著

RT-Thread 源代码的组织结构



bsp: 系统支持的处理器板级支持包(版本更新会逐渐添加新的处理器)

include: 系统最顶层的配置头文件

libcpu: 与移植有关的关键代码，支持的处理器关键移植代码在这里

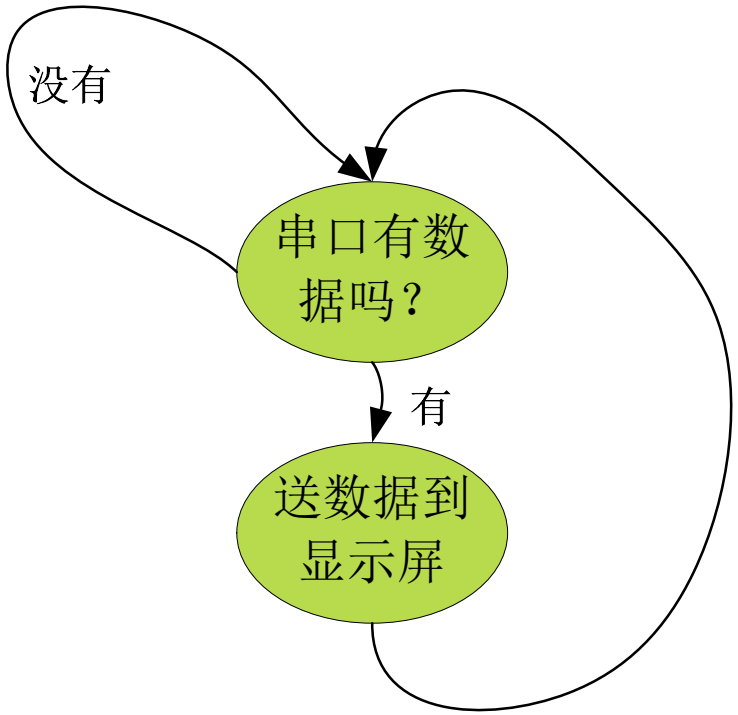
src: 整个系统核心代码

components: 内核之外的组件，不是必须，随着应用的深入会逐渐需要

为什么需要(嵌入式)操作系统(EOS)

- OS可以显著降低开发难度
- 让每一个任务都认为自己独占CPU, 方便代码编写
- 增加代码的移植性

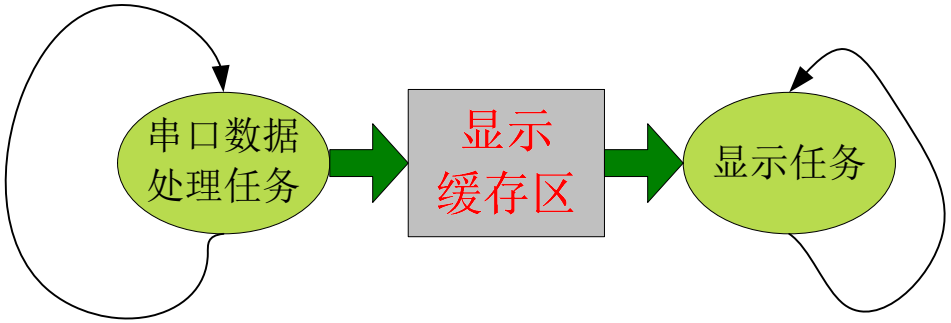
• 裸机系统



任务无休止查询.

缺点：工程复杂度增加时，带来程序编写的困难和实时性问题！

•带OS(多线程系统)



多个线程互相协作.

优点：工程复杂时只需添加更多的线程，实时性通过合理规划任务的优先级能够得到保证！

实时性：在规定的时间内响应

- 裸机多任务轮询系统示例

```
While(1)
{
  if( uart == 1 )
    { //串口任务:串口接收数据并在LCD显示器上显示
      read_uart( );
      write_to_lcd( );
    }
  If( timer == 1 )
   { //定时器任务
   }
  If( tcpip == 1)
   { //网络任务
   }
}
```

多线程系统示例

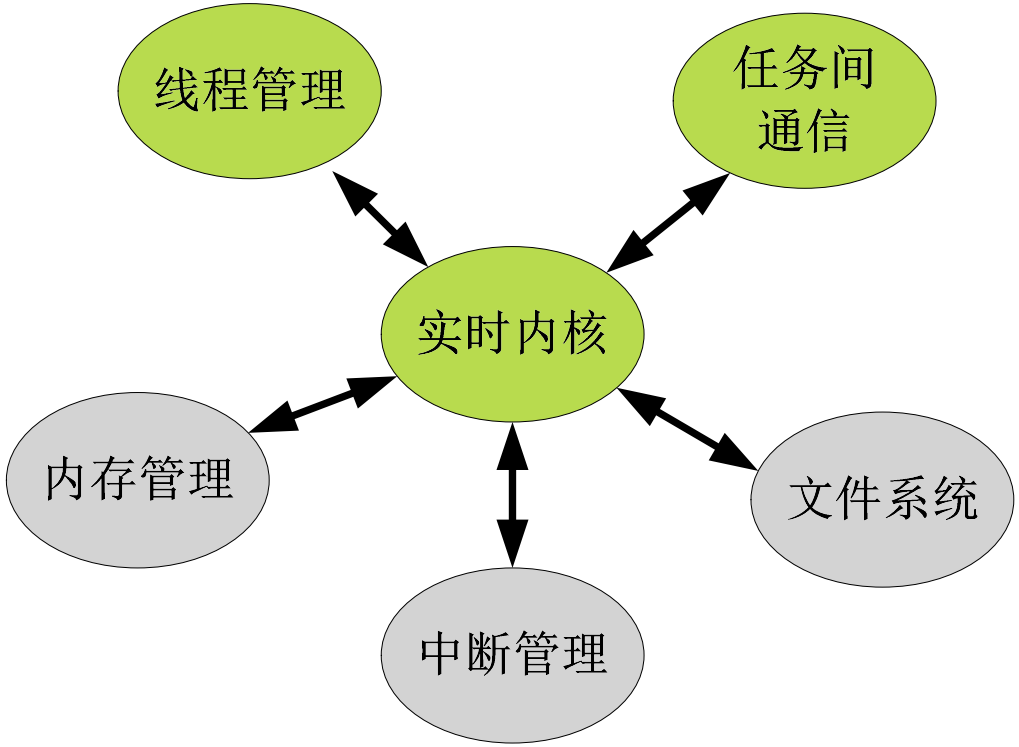
```
int main(void)
{  HardWareInit(); /*硬件初始化*/
   RTOSInit();       /*OS初始化*/
   RTOSStart();     /*OS启动，开始多线程调度，不再返回*/
}

/*串口接收线程*/
void UartThread(void)
 {  read_usart();  }

/*液晶显示线程*/
void LcdThread(void)
 { write_to_lcd(); }

/*定时器处理线程*/
void TimerThread(void)
 { ; }
```

何为线程？

具有不同功能、完成不同任务的函数。

EOS的几个关键问题

- 实时内核
- 线程管理
- 任务间通信
- (中断管理)
- (内存管理)
- (文件系统)



❖ 实时内核(Real Time Kernel)

负责管理多个线程的调度、任务间的通信与同步、存储器管理及中断管理。

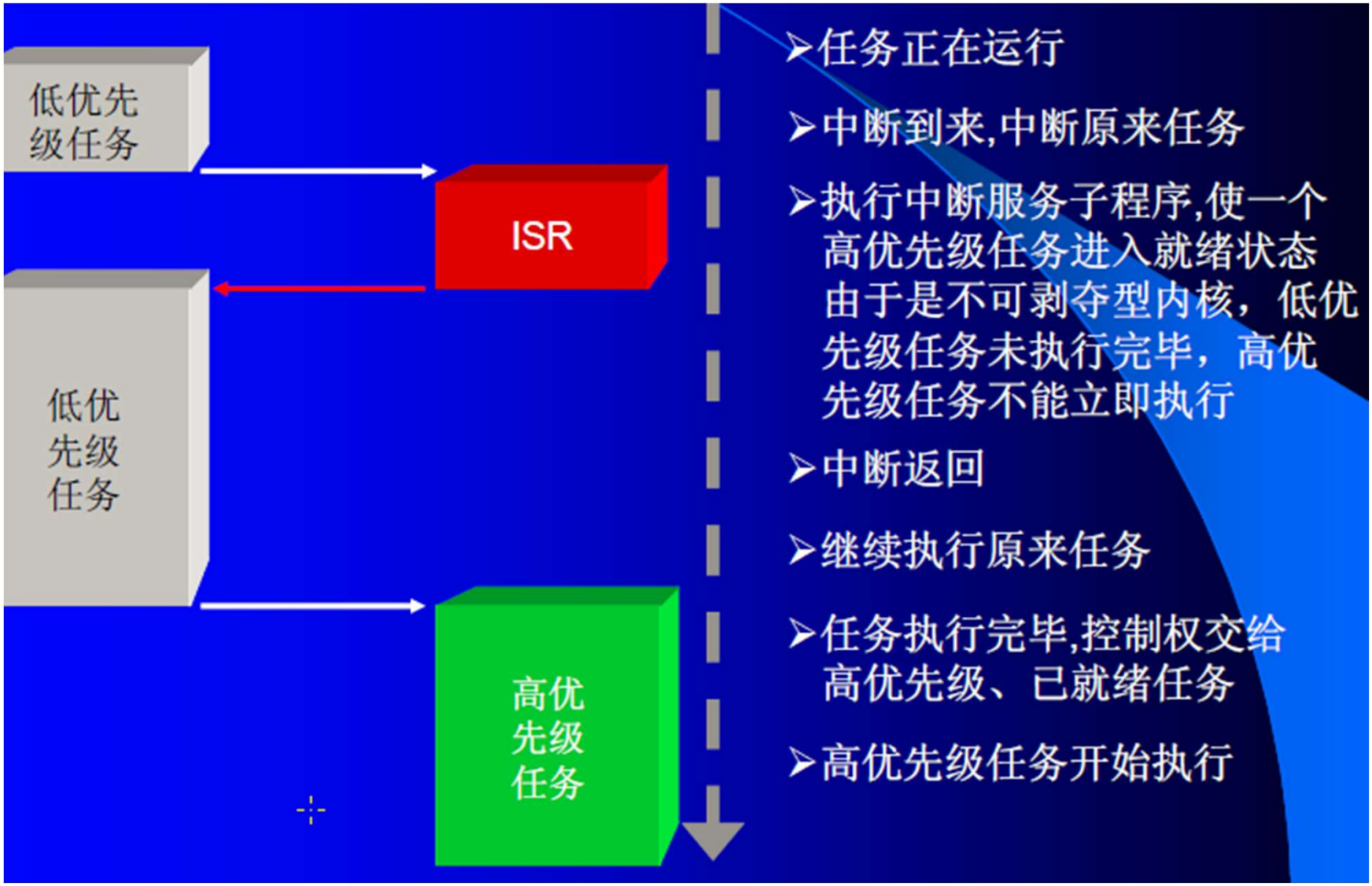
- 不可剥夺型内核

正在运行的任务不可以被更高优先级的任务剥夺CPU的使用权。

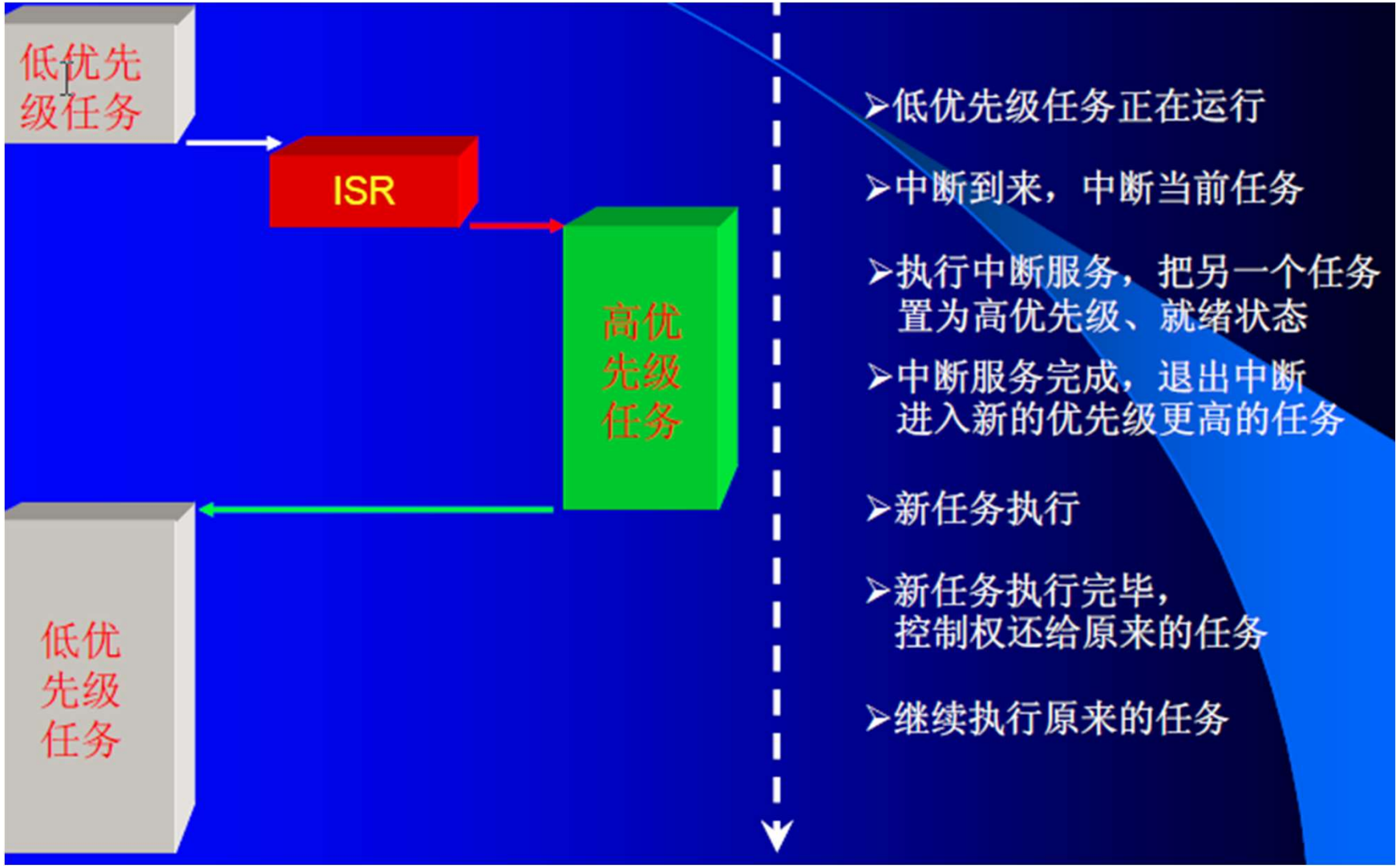
- 可剥夺型内核

正在运行的任务可以被更高优先级的任务剥夺CPU的使用权。

不可剥夺型内核图示

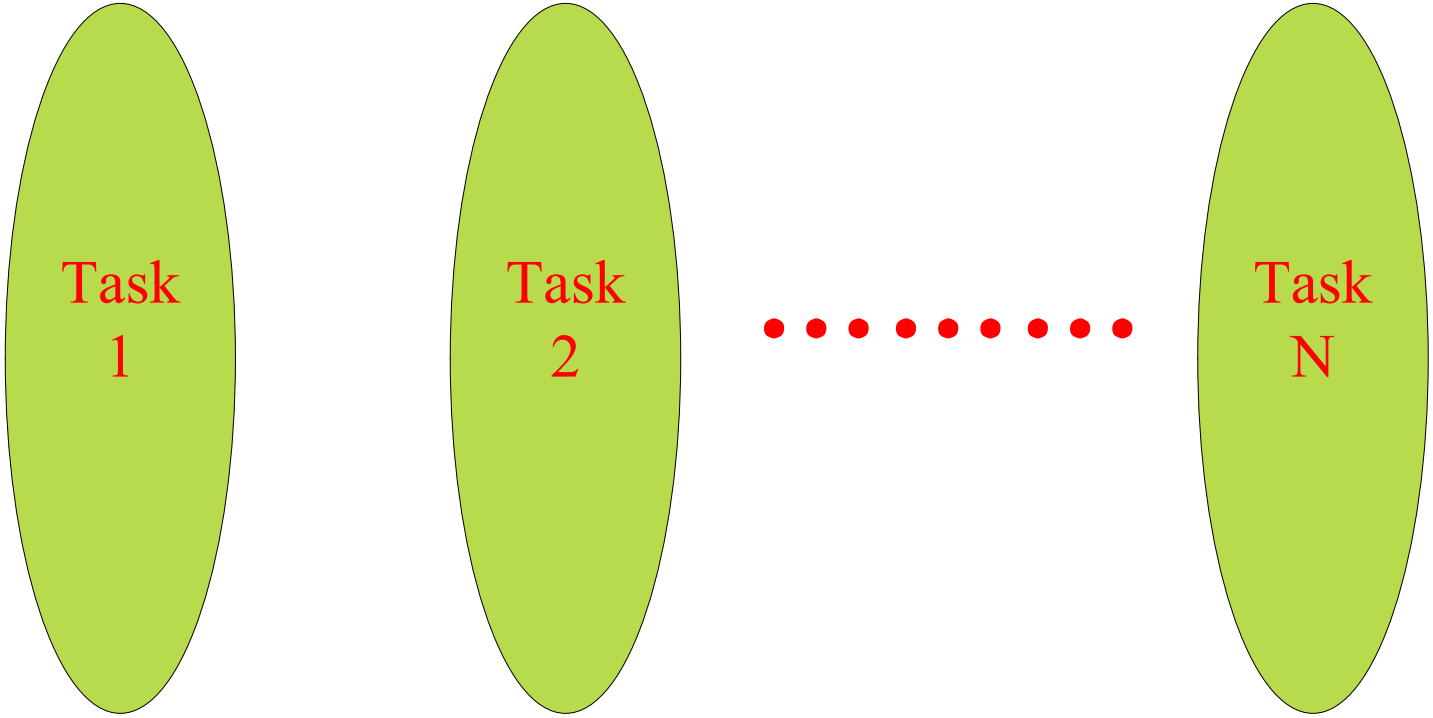

低优先级任务
ISR
低优先级任务
高优先级任务
任务正在运行
中断到来,中断原来任务
执行中断服务子程序,使一个高优先级任务进入就绪状态
由于是不可剥夺型内核，低优先级任务未执行完毕，高优先级任务不能立即执行
中断返回
继续执行原来任务
任务执行完毕,控制权交给高优先级、已就绪任务
高优先级任务开始执行

可剥夺型内核(RT-Thread 是可剥夺型内核)



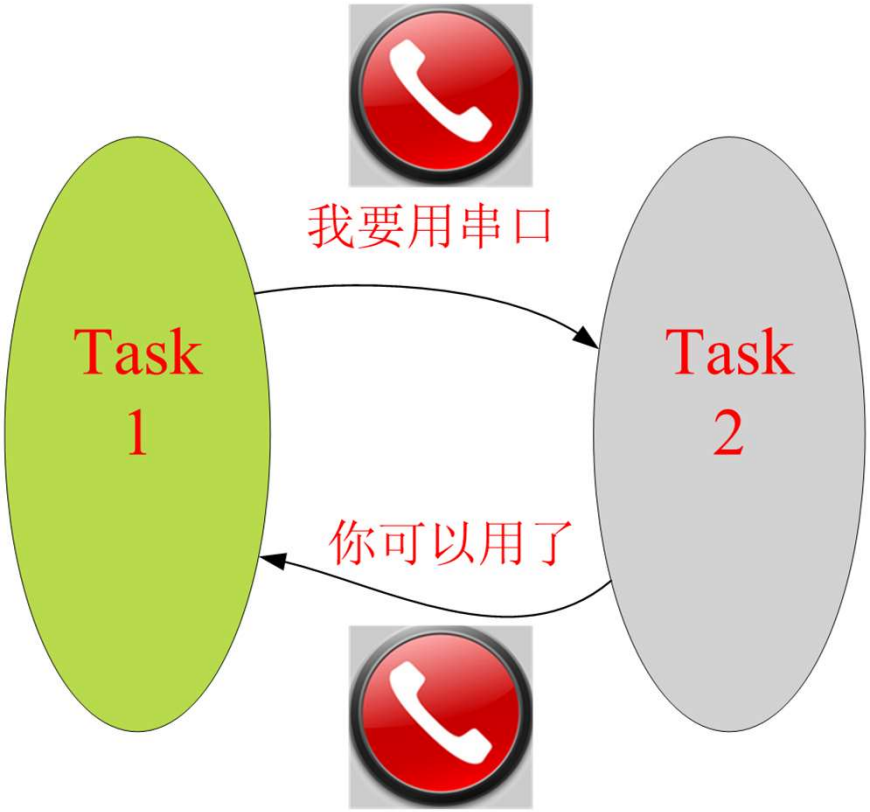
❖ 线程管理

需要哪些任务线程？
哪一个要占用CPU？



线程间通信

功能：保证线程任务之间的良好协作



需要采用什么通信机制？

2 线程调度与管理

❖ 实时系统需求

- 实时性指固定时间内对外部事物作出响应

不一定是马上响应，
固定时间内响应就可以

- 实时系统是需求倾向性系统和等级系统

  实时重要任务第一时间作出回应，非实时性任务则为之让路。

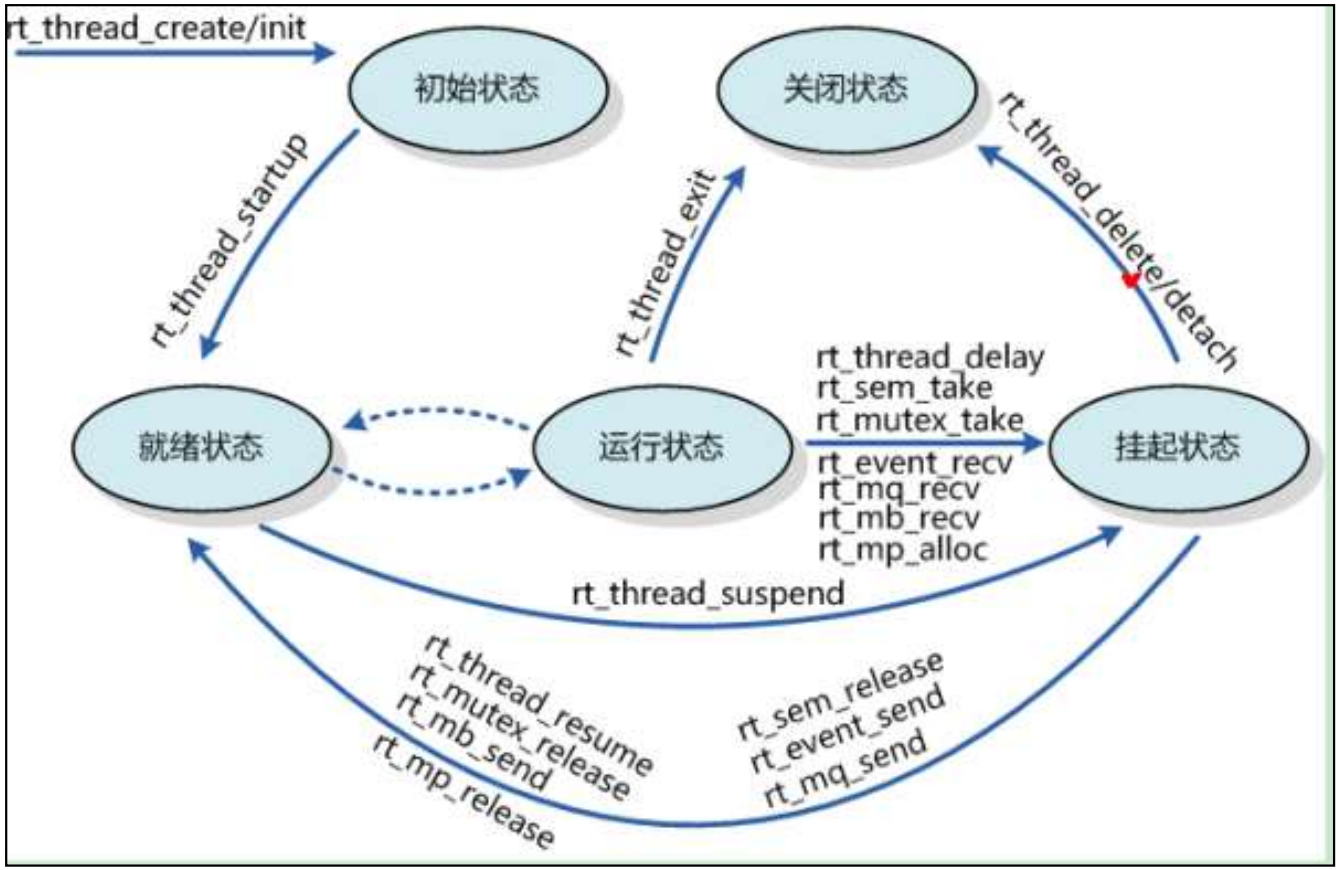
  举例：用手机听音乐、刷微信时有电话进来, 系统是如何反应的？

- **RT-Thread**中，任务采用线程实现，线程是最基本任务调度单位。

线程状态

- 线程的5种状态

——初始化, 就绪, 运行, 挂起(睡眠), 关闭



线程转换图(箭头旁为转换调用函数)

- 初始化状态

<Tip:长函数抓关键词>

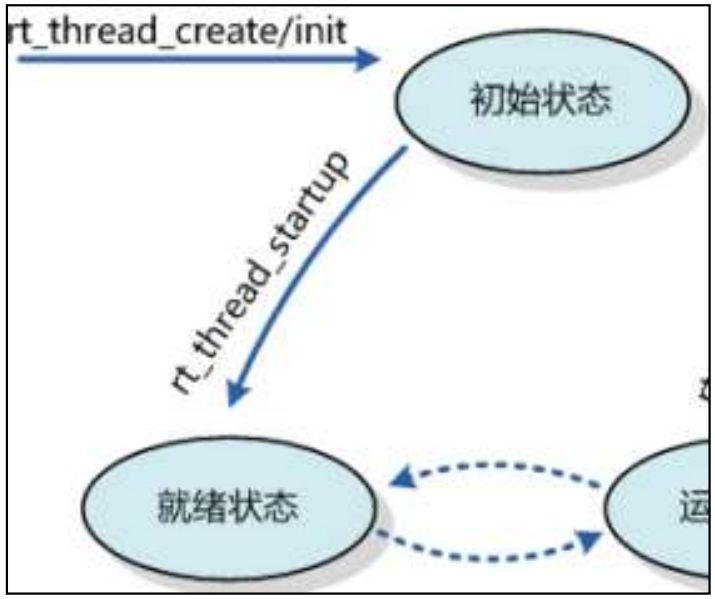
  - 调用rt_thread_init函数实现
  - 示例:初始化一个LED1管理线程(详见application.c)

```
/*需先定义一个线程控制块结构体*/
 struct rt_thread Thread_Handle_Led1Mgmt;
/*调用init函数完成初始化*/
 rt_thread_init(&Thread_Handle_Led1Mgmt, //加载
                                         //线程结构体
                Thread_entry_Led1Mgmt, //处理函数入口
                THREAD_LED1_MGMT_PRIO, //优先级
                10,  //时间片，同级线程轮转时间
                ..... ) //其他参数，略
```

就绪状态

- 调用**rt_thread_startup**函数
- 示例:启动**LED1**线程进入就绪状态

```
rt_thread_startup(&Thread_Handle_Led1Mgmt);
```

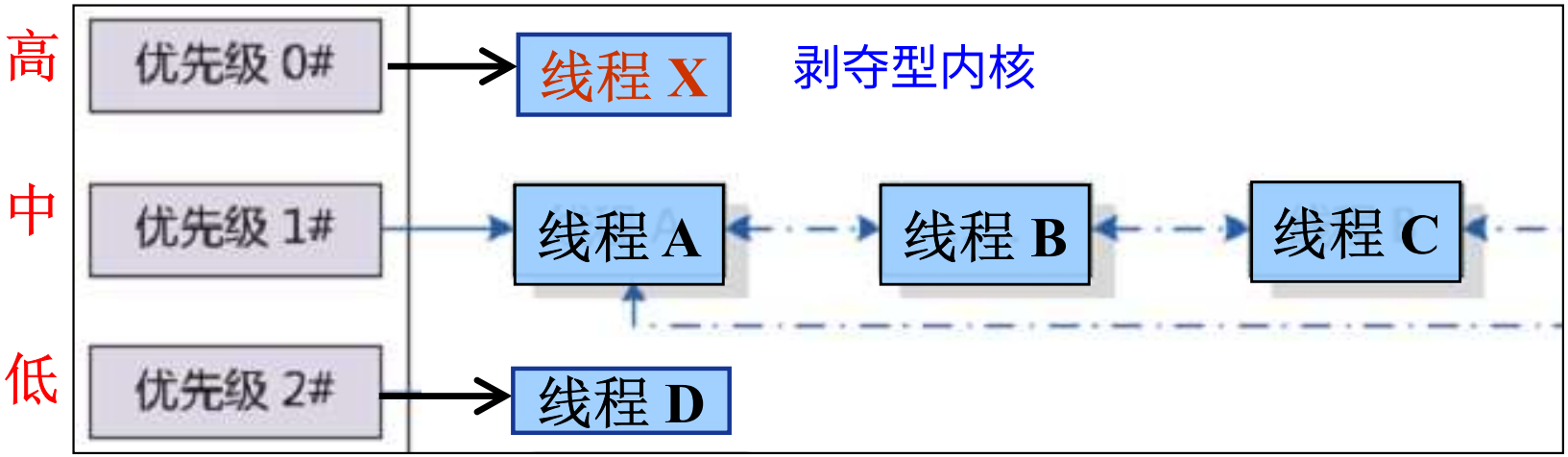


线程从初始态到就绪态的转换

❖ 运行状态

- 就绪态到运行态无需调用函数，只需等待机会

- 线程就绪后即进入线程队列等待调度(下图)



同级线程(A,B,C)轮转运行, 时长由时间片决定.

高级线程(X)可抢占中级线程运行权限. (示意图)

低级线程D何时能运行?

❖ 挂起(睡眠)状态

- 可调用**rt_thread_delay(time)**

- 高级别线程全部挂起后, 低级别才获得运行权限

  前例: 线程X A B C全挂起后, 线程D才可运行.

- 调用位置

  在线程处理函数中调用.

线程处理函数

- 线程运行时, 完成具体任务的函数, 由用户设计.
- 示例: LED1线程的处理函数

```
void Thread_entry_Led1Mgmt()  //需事先声明
{
  while(1)
  {
        printf("线程1运行, LED1闪烁\n");
        LedToggle(LED1);    // 线程任务: 切换一次灯态
        rt_thread_delay(20); // 挂起20个时间单位
  }
}
```

剥夺对CPU的使用权，如果删除不挂起，低级任务永远没有使用权。

Tips 线程函数是一个while循环，永不返回!

谈谈时间单位

重要

- 1个时间单位称：

  1次系统滴答(SysTick)或1次OS的心跳，它是线程运行或挂起的时间基准.

- 1次滴答是多长时间？

  在rtconfig.h中定义了每秒的滴答数：

  #define RT_TICK_PER_SECOND　　100; //100次/秒

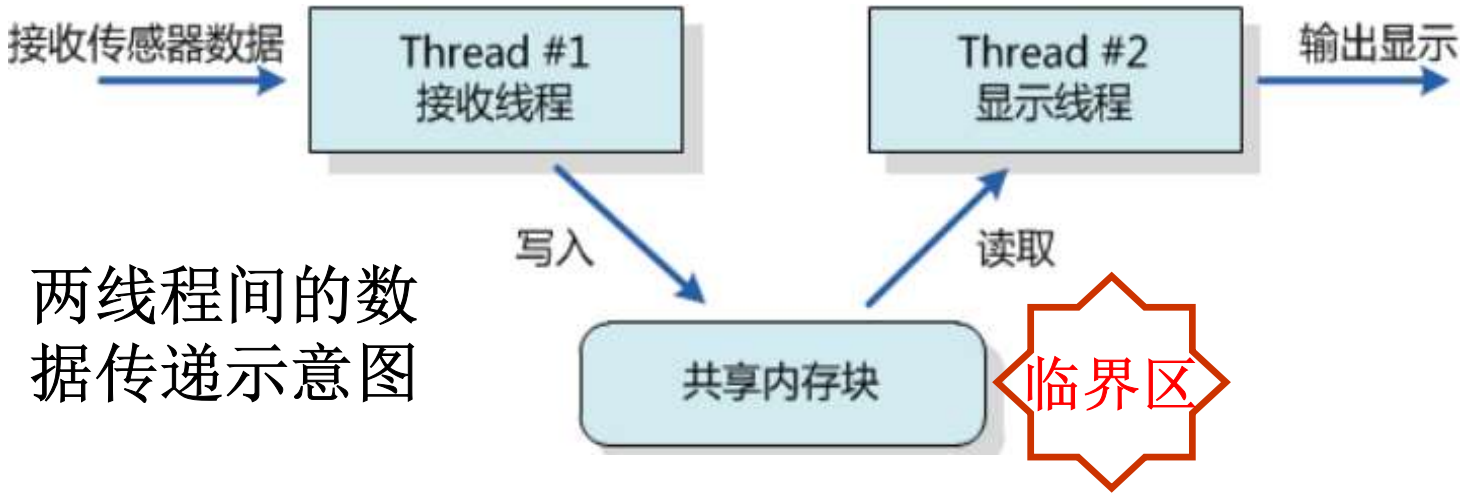
  则每次滴答(心跳)的时间为: 10ms, 由系统滴答定时器提供定时.

- 前例: rt_thread_delay(20) 挂起时间为200ms.

3 任务间同步与通信

❖ 同步及通信的需求

- 一项工作往往需要多个线程以协调方式完成(下图).
- 两线程访问/操作的同一块区域称临界区.
- 同时访问临界区会出现数据一致性问题.
- 任务同步核心思想: 访问临界区只允许一个(或一类)任务运行.



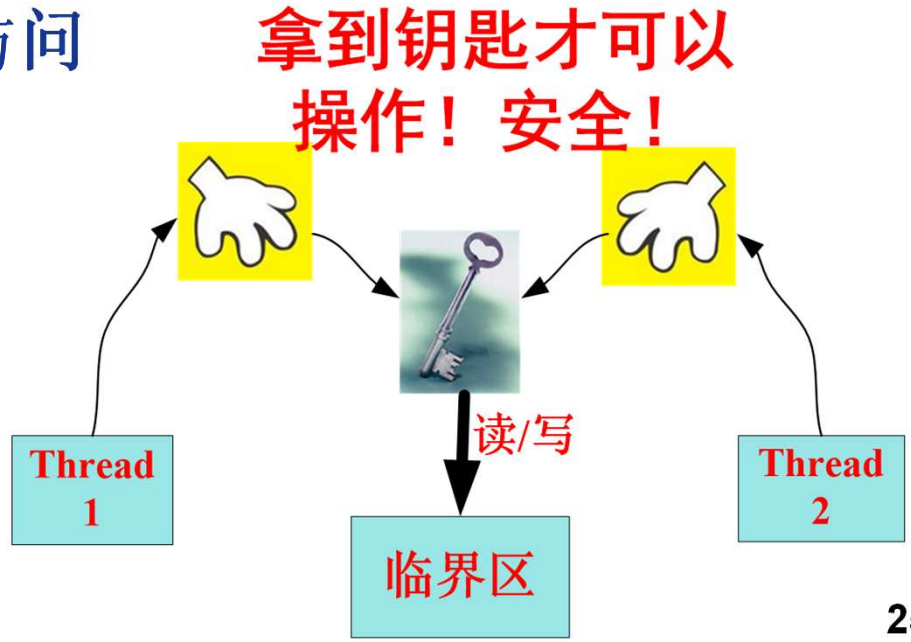
两线程间的数据传递示意图

任务间通信的机制

5种: 信号量, 事件, 消息队列, 邮箱, 互斥量

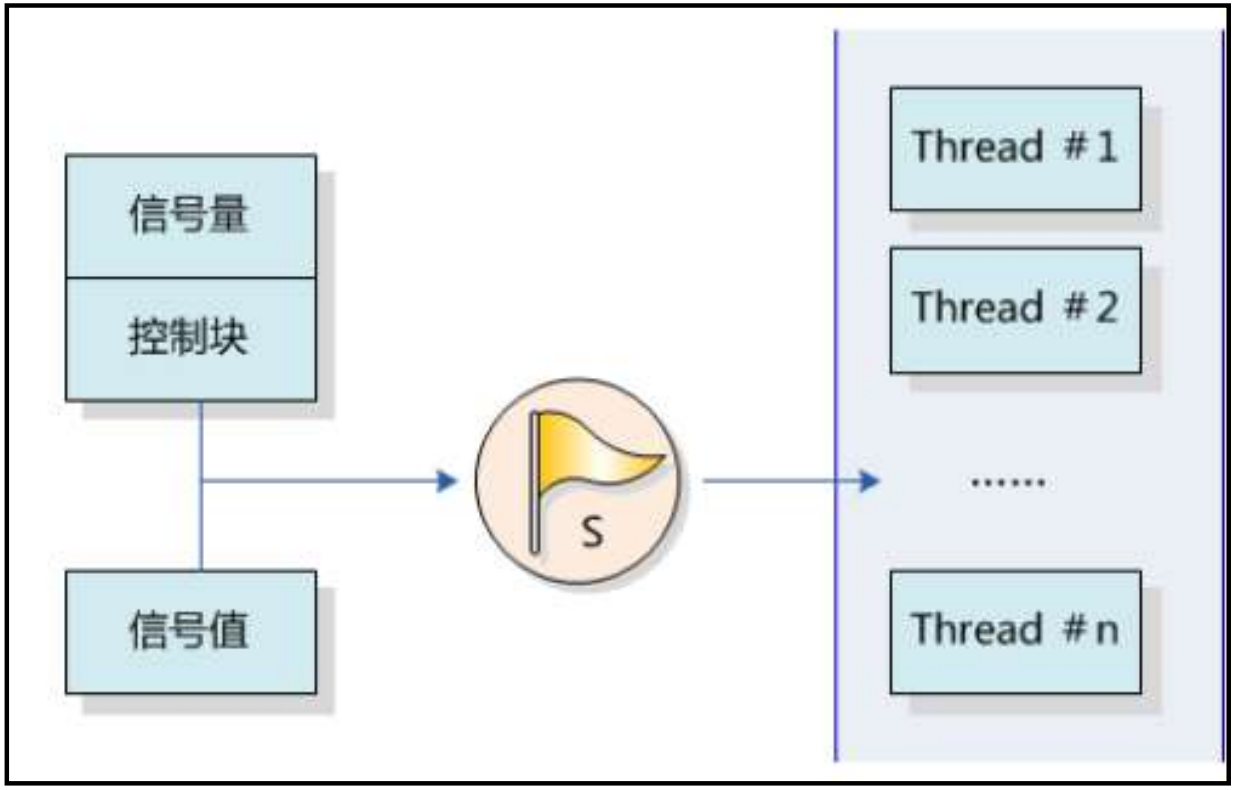
信号量(Semaphore)

- 一种解决线程同步问题的轻型通信机制.
- 等同于打开临界区的一把钥匙(如图).
- 获取到信号量的线程才可访问临界区, 用后释放; 另一线程则需等待.



- 信号量工作示意

  信号量对象包括:信号值(资源数/钥匙数)和等待的线程队列



信号量工作示意图

信号量接口函数

- /*声明一个信号量*/

```
struct rt_semaphore  SemUser;
```

- /*信号量初始化函数原型*/

```
rt_sem_init ( rt_sem_t    sem,    //待初始化的信号量变量
              const char  *name,  //信号量的名字(类似人名)
              rt_uint32_t  value,  //信号量初值(钥匙数)
              rt_uint8_t    flag,   //信号量的处理方式标志
              )
```

- 初始化示例：

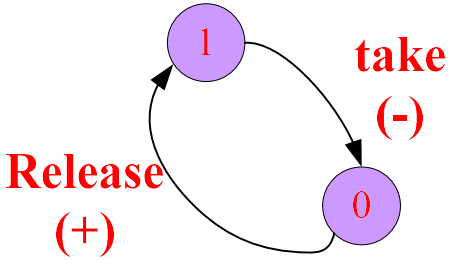
```
rt_sem_init(&SemUser , "SemUser", 1 , …_FIFO);
/*FIFO(First In First Out), 按排队顺序拿钥匙,
                          而不是按线程优先级.*/
```

- 信号量获取(take)和释放(release)函数

```
rt_sem_take(rt_sem_t sem,   //想要拿的信号量
            rt_int32_t time,  //拿不到, 等待的Tick数
            )
rt_sem_release(rt_sem_t sem) //要释放的信号量
```

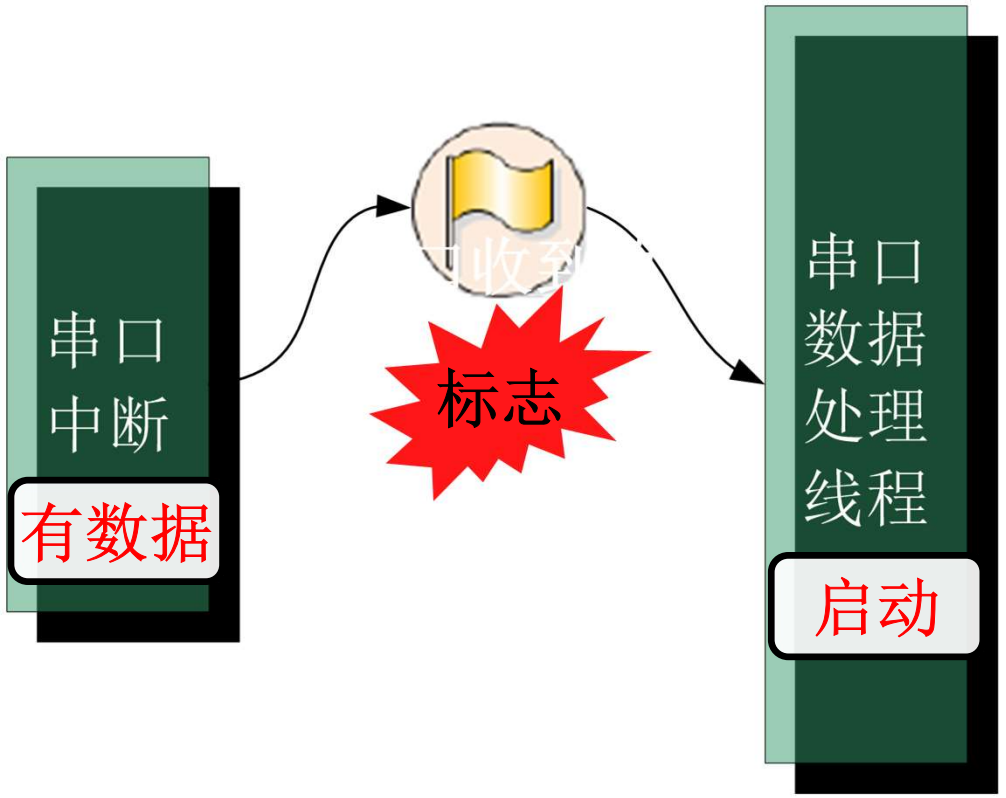
- Take一次计数器减1, release一次加1(如图);
  为0时take不到, 则线程挂起.

信号量计数器
的变化过程图

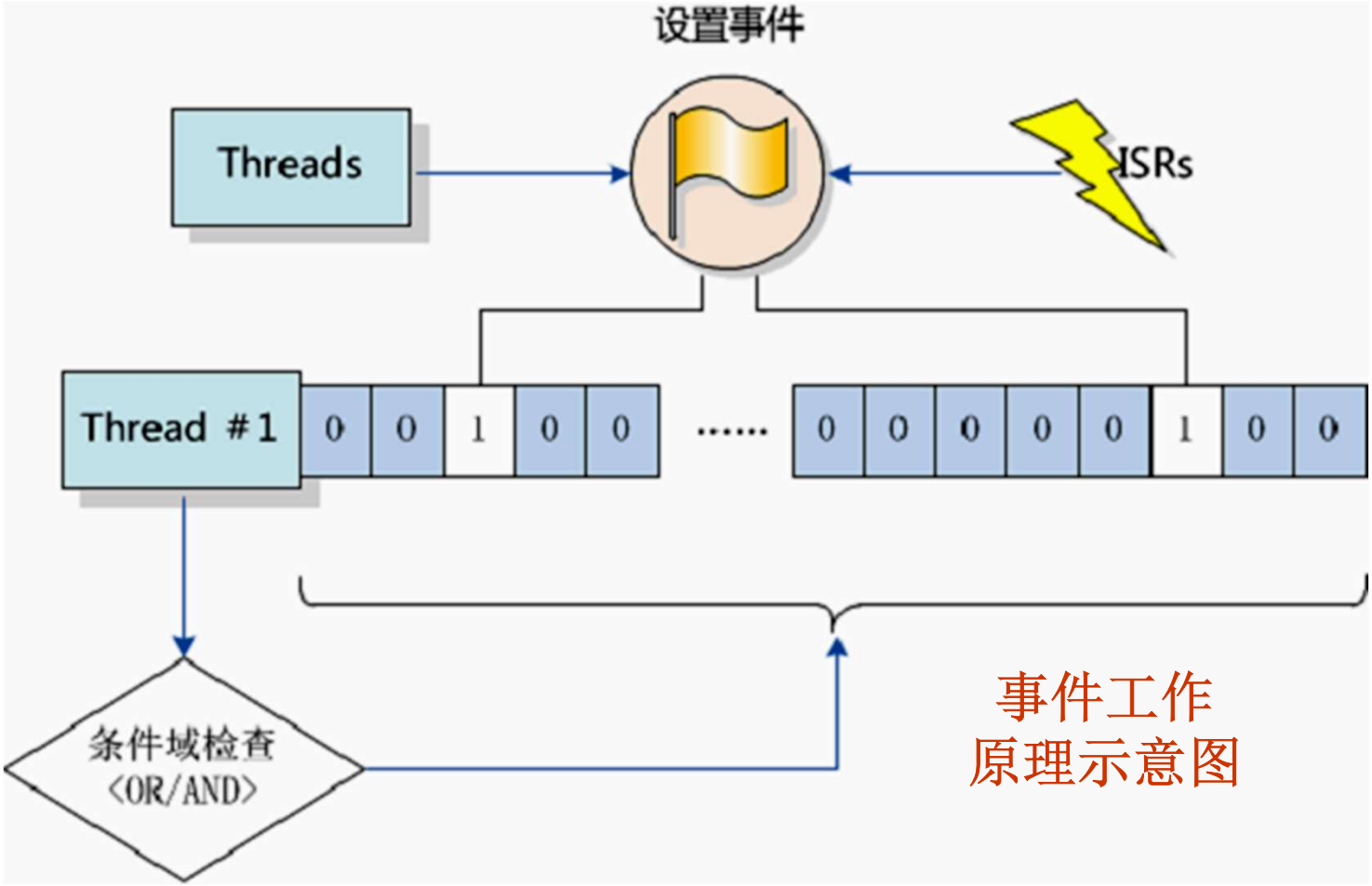


❖ 通信机制——事件(Event)

- 事件发生设置标志位, 用于实现线程同步.
- 示例如下图:串口接收数据后中断, 设置事件标志, 而后处理线程根据标志运行.



- 事件可实现一对多, 或多对多线程同步.
- 事件变量为32位整型, 有32个标志位 (如0x00000001,…)
- 事件仅用于同步, 并不传输数据.



事件工作
原理示意图

事件函数及应用举例

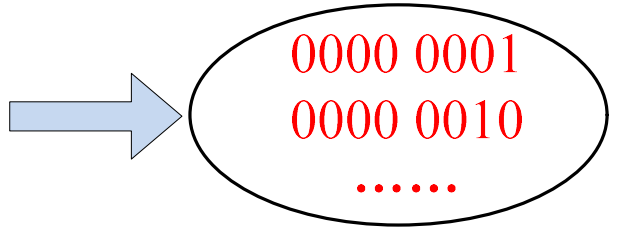
- /*声明一个事件变量*/

```
struct rt_event   BeepEvent; //蜂鸣事件
```

- /*定义具体事件及标志位*/

```
#define EVENT_BEEP_ON    0x01 //响
#define EVENT_BEEP_OFF 0x02 //关
```

位码形式最多
32个具体事件



调用事件事为1的比特位
不能重叠

- /*蜂鸣事件初始化*/

```
rt_event_init(&BeepEvent , "BeepEvent", …_FIFO);
```

//3个参数分别为:事件变量,　事件名字,　处理方式//

❖ 事件设置线程和事件处理线程示例

蜂鸣事件设置线程(伪代码)

```
while(1)
 {
   查询按键情况;
   K1按下：
   send event (EVENT_BEEP_ON);
   K2按下：
   send event (EVENT_BEEP_OFF);
   线程睡眠20ms;
 }
```

系统负责事件排队及任务调度

蜂鸣器控制线程

```
while(1)
 {
   recveive(EVENT);
   处理蜂鸣器事件;
 }
```

事件发送和接收函数示例

- /*设置蜂鸣响标志位, 并发送蜂鸣事件*/

```
rt_event_send(&BeepEvent , EVENT_BEEP_ON);
```

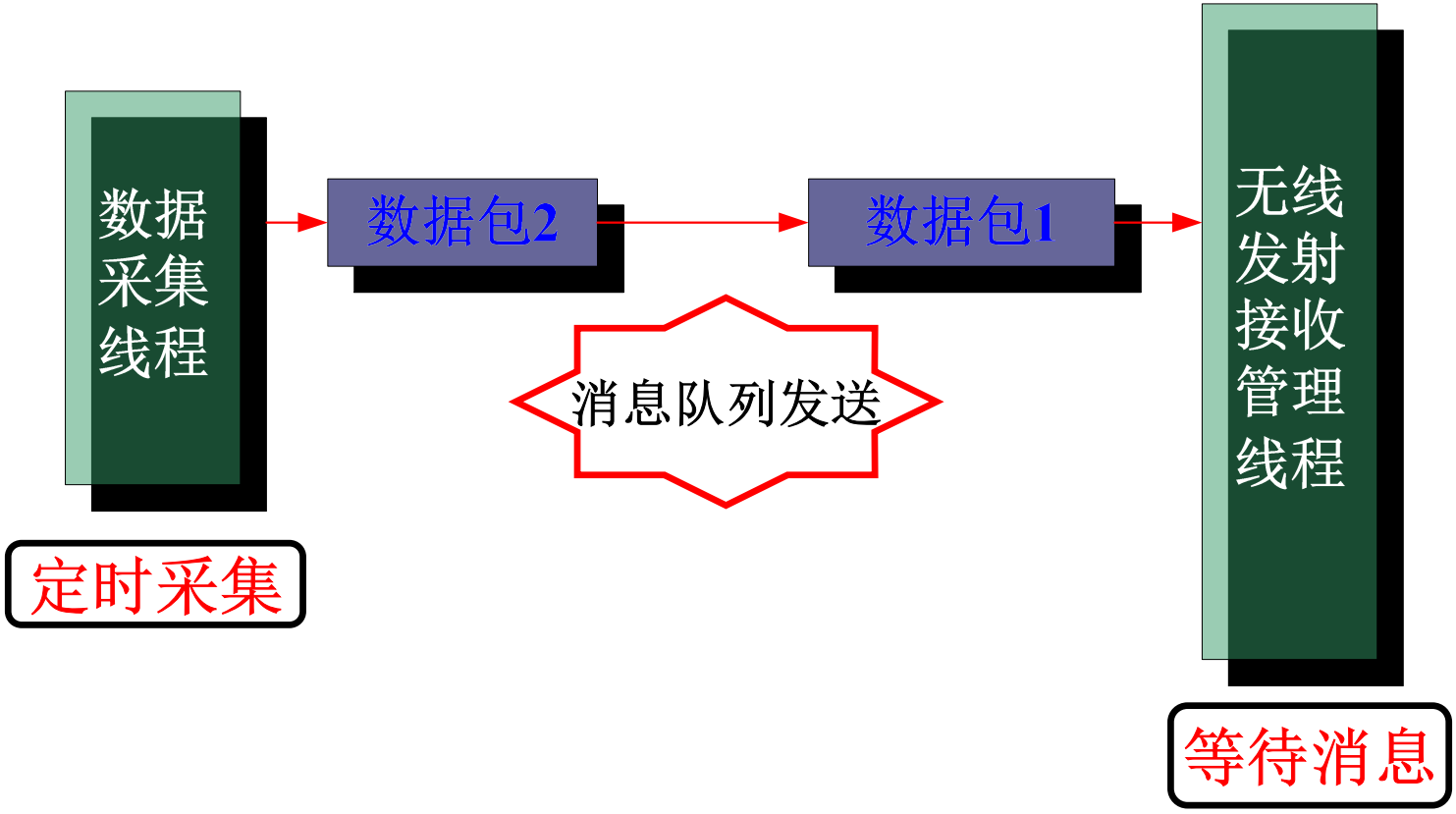
- /*接收蜂鸣事件*/

非Check

```
rt_event_recv(&BeepEvent, //事件变量
              //要检查的事件标志
              EVENT_BEEP_ON | EVENT_BEEP_OFF,
              //检查条件：逻辑或
              RT_EVENT_FLAG_OR .....);
```

通信机制——消息(Message)

- 常用的线程通信方式, 消息缓存在自己的内存.
- 消息队列长度不固定, 便于一个线程或中断ISR将特定数据发给另一线程(示意图如下).



- 消息接口函数
  - /*声明一个消息变量及消息长度*/

```
struct rt_messagequeue  UsartMsg; //串口消息
rt_uint8_t   UsartMsgPool[16]; //16字节
```

  - /*消息初始化*/

```
rt_mg_init (&UsartMsg, “UsartMsg",UsartMsgPool…)
//3个主参分别为:消息变量，  消息名字，  消息缓冲区//
```

消息发送和接收函数示例

- /*消息发送*/

```
u8 data;
data = USART_ReceiveData (USART1); //从串口1获取一个数据
//将串口收到的数据data作为消息UsartMsg发送//
rt_mg_send (&UsartMsg , &data, ……);
```

- /*消息接收*/

```
char msg
//从消息队列UsartMsg接收消息并保存到msg//
rt_mg_recv (&UsartMsg , &msg , ……);
```

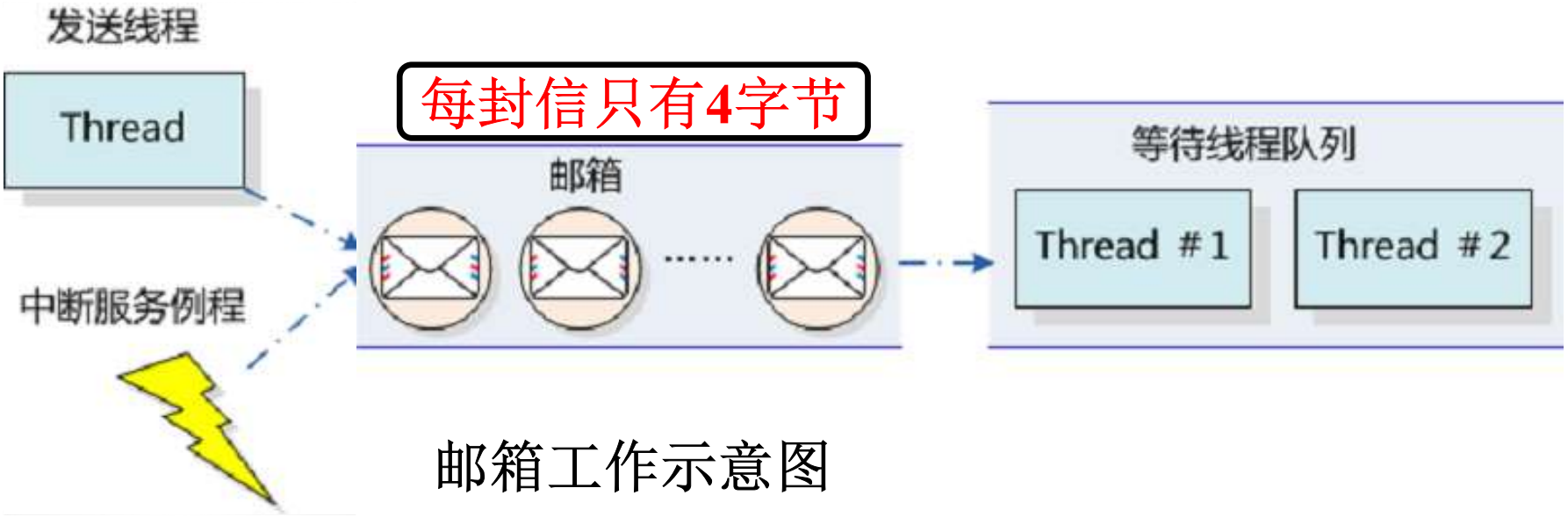
❖ 前例数据和消息传递路径示意图

思考：事件发送和消息发送有何不同？

- 事件发送的仅是标志位，而非数据；
- 消息发送的是数据本身。

❖ 通信机制——邮箱(MailBox)

- 邮箱用来在线程之间传递邮件(Mail).
- 邮件可以看成消息的一种特殊形式, 其长度只有4字节.
- STM32中,一个指针变量(地址)刚好32位, 利用邮箱可以传送指针.



邮箱工作示意图

思考题

- 带嵌入式操作系统的编程有什么优势？
- 什么是线程？
- 线程的状态有哪些？如何相互转换？
- 简要解释线程通信方式中的信号量、事件和消息