What is uC/OS?

u: Micro C:control

uC/OS : 适合于小的、控制器的操作系统

- 小巧
- 公开源代码，详细的注解
- 可剥夺实时内核
- 可移植性强
- **多任务**
- 确定性

The Story of uC/OS

- 美国人Jean Labrosse 1992年编写的
- 商业软件的昂贵
- 应用面覆盖了诸多领域，如照相机、医疗器械、音响设备、发动机控制、高速公路电话系统、自动提款机等

嵌入式操作系统—uC/OS

- **μCOS不但提供了一个完整的嵌入式实时内核的源代码，而且对这些代码的细节作了详尽的解释，它不仅告诉读者这个实时内核是怎么写的，还解释了为什么要这样写。而商业上的实时操作系统软件不但价格昂贵（一般都在5千到2万美元的价位上），而且其中很多都是所谓黑盒子，即不提供源代码。**
- **l 源代码的绝大部分是用C语言写的，经过简单的编译，读者就能在PC机上运行，边读书、边实践。由于用汇编语言写的部分只有200行左右，该实时内核可以方便地移植到几乎所有的嵌入式应用类CPU上。移植范例的源代码可以从因特网上下载。**
- **l 从最老版本的实时内核μCOS，以及后来的μC/OS，到新版本的μC/OS-II，已经有多年的历史。许多行业上都有成功应用该实时源代码.实时内核移植.内核实时内核的实例，这些应用的实践是该内核实用性、无误性的最好证据。**

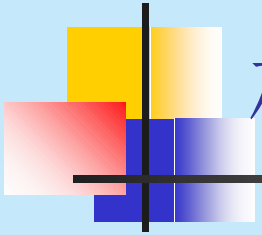
嵌入式操作系统—uC/OS

1）. NSA2010便携式电话，在日本大约有15000台投入市场。使用μC/OS实时操作系统。

2）. CYCLONE移动电话，Hitachi H8S/2318k微程序控制器，256K闪存和8K Ram， μC/OS 实时操作系统。

选择μC/OS的原因：

INFEA R&D的职员从1996年以来开始应用Micriμm实时操作系统。通过比较，还没有发现比μC/OS更好的实时操作系统。我们将继续应用μC/OS以及Micriμm的其它产品包括下一代μC/OS-II V2.52的产品。



概要

- 内核结构-任务以及调度机制
- 任务间通信
- uC/OS的移植
- 在PC机上运行uC/OS

任务task

- 典型的一个无限循环。

```
void mytask(void *pdata)
{
   for (;;) {
do something;
waiting;
do something;
   }
}
```

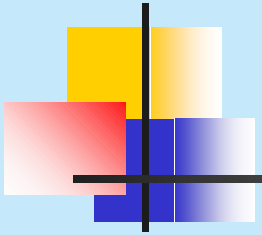
- 支持64个任务，每个任务一个特定的优先级。优先级越高，数字越小
- 系统占用了两个任务，空闲任务和统计任务。

任务优先级

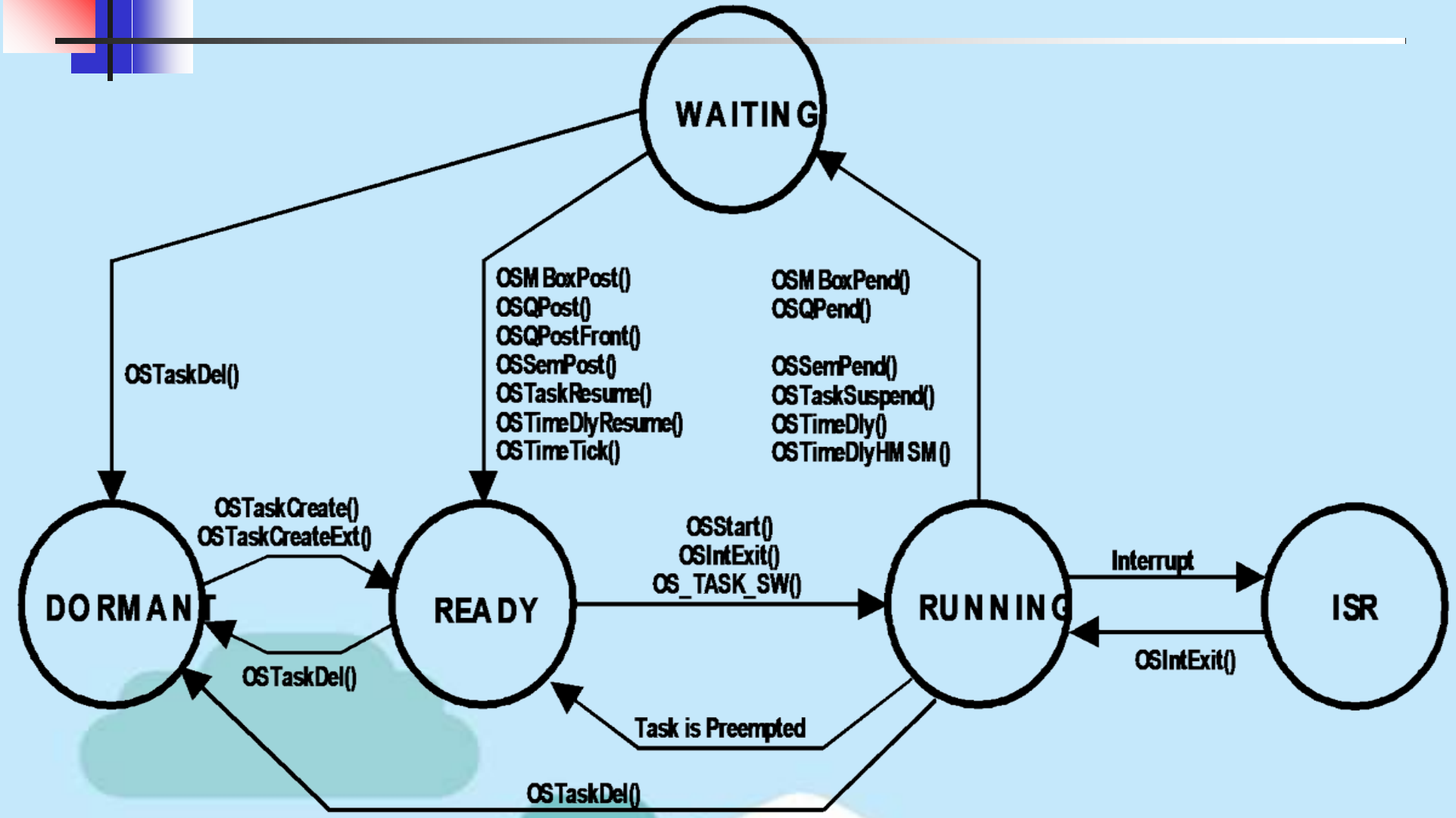
- 每个任务按其重要性被赋予一定的优先级。μC/OS-Ⅱ可以管理多达64个任务，但目前版本的μC/OS-Ⅱ有两个任务已经被系统占用了。作者保留了优先级为0、1、2、3、OS_LOWEST_PRIO-3、OS_LOWEST_PRI0-2，OS_LOWEST_PRI0-1以及OS_LOWEST_PRI0这8个任务以被将来使用。OS_LOWEST_PRI0是作为定义的常数在OS_CFG.H文件中用定义常数语句#define constant定义的。因此用户可以有多达56个应用任务。必须给每个任务赋以不同的优先级，优先级可以从0到OS_LOWEST_PR10-2。优先级号越低，任务的优先级越高。μC/OS-Ⅱ总是运行进入就绪态的优先级最高的任务。目前版本的μC/OS-Ⅱ中，任务的优先级号就是任务编号（ID）。优先级号（或任务的ID号）也被一些内核服务函数调用，如改变优先级函数OSTaskChangePrio()，以及任务删除函数OSTaskDel()。

任务状态

- 休眠态(dormant)：指任务驻留在程序空间中，还没有交给内核管理。把任务交给内核是通过调用OSTaskCreate( )或OSTaskCreatExt( )实现的。
- 就绪(Ready)：当任务一旦建立，这个任务就处于就绪态准备运行。任务可以动态的被另一个程序建立，也可以在系统运行开始之前建立。如果一个任务是被另一个任务建立的，而这个任务的优先级高于建立它的那个任务，则这个刚刚建立的任务将立即得到CPU的控制权。通过调用OSTaskDel( )使任务返回到休眠态。就绪态的任务都放在就绪列表中。在任务调度时，指针OSTCBHighRdy指向优先级最高的就绪任务，也就是立刻就要运行的任务。



- 运行(Running)：准备就绪的最高优先级的任务获得CPU的控制权，从而处于运行态。指针OSTCBCur指向正在运行的任务。
- 等待或挂起(Pending)：正在运行的任务由于调用延时函数OSTimeDly( )或等待事件信号量的来临而将自身挂起，因而处于等待或挂起态。因为等待某事件而被挂起的任务注册在该事件的等待列表中。
- 中断态(Interrupt)：正在运行的任务可以被中断，除非是该任务将中断关闭。被中断的任务进入中断服务程序(ISR)。如果中断服务程序使一个更高优先级的任务准备就绪，则中断服务程序结束后，更高优先级的任务开始运行程序。


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

任务堆栈

- 在μC/OS-Ⅱ中，每个任务都有自己的堆栈空间。为方便使用，在μC/OS-Ⅱ中专门定义了一个OS_STK类型的数据，这样在应用程序中定义任务的堆栈就非常方便。例如：
- 程序 8-5
- #define TASK_STK_SIZE 200
- OS_STK TaskStartStk[TASK_STK_SIZE];

任务的数据结构—任务控制块

- 任务控制块 OS_tcb，包括

  任务堆栈指针，状态，优先级，任务表位置，任务链表指针等。

- 所有的任务控制块分为两条链表，空闲链表和使用链表。

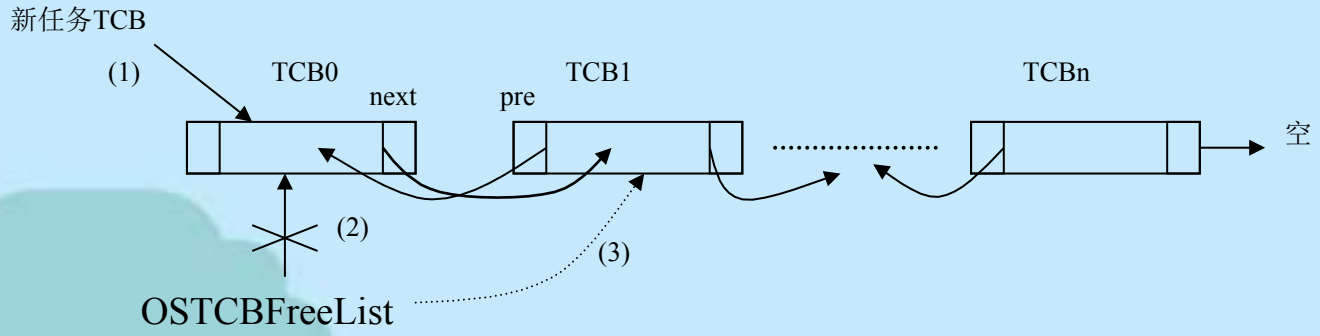


图 4.3 TCB的双向链表结构

任务控制块结构

- Struct os_tcb {

```
    OS_STK       *OSTCBStkPtr;
   struct os_tcb *OSTCBNext;
  struct os_tcb *OSTCBprev;
  OS_EVENT    *OSTCBEventPtr;
   void           *OSTCBMsg;
    INT16U    OSTCBDly;
    INT8U      OSTCBStat;
  INT8U     OSTCBPrio;
    INT8U       OSTCBX, OSTCBY, OSTCBBitX, OSTCBBitY;
} OS_TCB
```

- **OSTCBStkPtr**是指向当前任务栈顶的指针。μC/OS-Ⅱ允许每个任务有自己的栈，尤为重要的是，每个任务的栈的容量可以是任意的。有些商业内核要求所有任务栈的容量都一样，除非用户写一个复杂的接口函数来改变之。这种限制浪费了RAM，当各任务需要的栈空间不同时，也得按任务中预期栈容量需求最多的来分配栈空间。OSTCBStkPtr是OS_TCB数据结构中唯一的一个能用汇编语言来处置的变量（在任务切换段的代码Context-switching code之中），把OSTCBStkPtr放在数据结构的最前面，使得从汇编语言中处理这个变量时较为容易。

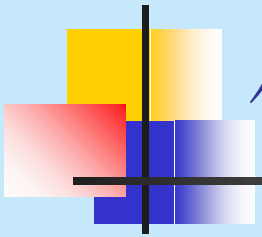
- **.OSTCBNext**和**.OSTCBPrev**用于任务控制块OS_TCBs的双重链接，该链表在时钟节拍函数OSTimeTick()中使用，用于刷新各个任务的任务延迟变量.OSTCBDly，每个任务的任务控制块OS_TCB在任务建立的时候被链接到链表中，在任务删除的时候从链表中被删除。双重连接的链表使得任一成员都能被快速插入或删除。

- **OSTCBEventPtr**是指向事件控制块的指针，后面的章节中会有所描述（见8.9 任务的同步和通信）。
- **.OSTCBMsg**是指向传给任务的消息的指针。用法将在后面的章节中提到（见8.9 任务的同步和通信）。
- **.OSTCBDly**当需要把任务延时若干时钟节拍时要用到这个变量，或者需要把任务挂起一段时间以等待某事件的发生，这种等待是有超时限制的。在这种情况下，这个变量保存的是任务允许等待事件发生的最多时钟节拍数。如果这个变量为0，表示任务不延时，或者表示等待事件发生的时间没有限制

- **OSTCBStat是任务的状态字。当.OSTCBStat为0，任务进入就绪态。可以给.OSTCBStat赋其它的值，在文件uCOS_II.H中有关于这个值的描述。**
- **.OSTCBPrio是任务优先级。高优先级任务的.OSTCBPrio值小。也就是说，这个值越小，任务的优先级越高。**
- **.OSTCBX, .OSTCBY, .OSTCBBitX和 .OSTCBBitY用于加速任务进入就绪态的过程或进入等待事件发生状态的过程（避免在运行中去计算这些值）。这些值是在任务建立时算好的，或者是在改变任务优先级时算出的。**
- **.OSTCBDelReq是一个布尔量，用于表示该任务是否需要删除。**

任务的调度--OSSched

- uC/OS是占先式实时多任务内核，优先级最高的任务一旦准备就绪，则拥有CPU的所有权开始投入运行。
- uC/OS中不支持时间片轮转法，每个任务的优先级要求不一样且是唯一的，所以任务调度的工作就是：查找准备就绪的最高优先级的任务并进行上下文切换。



任务的调度

- 就绪任务表：用于存贮每个任务的就绪状态标志。由两个变量组成：
  - OSRdyGrp：8位，每位表示一组（8个）任务中是否有就绪的任务。
  - OSRdyTbl[]：位图方式表示某个任务是否就绪。

就绪状态标志

Bit 0 in OSRdyGrp is 1 when any bit in OSRdyTbl[0] is 1.

Bit 1 in OSRdyGrp is 1 when any bit in OSRdyTbl[1] is 1.

Bit 2 in OSRdyGrp is 1 when any bit in OSRdyTbl[2] is 1.

Bit 3 in OSRdyGrp is 1 when any bit in OSRdyTbl[3] is 1.

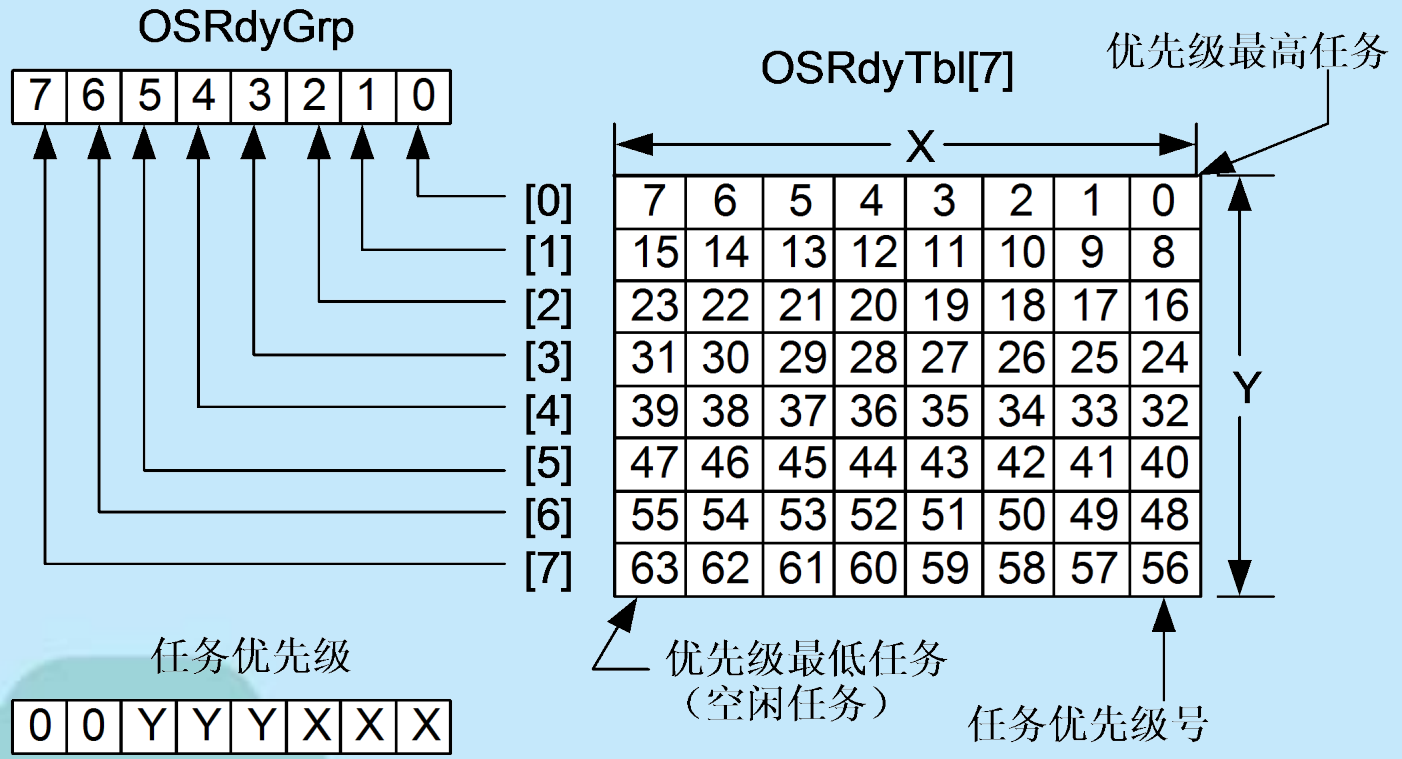
Bit 4 in OSRdyGrp is 1 when any bit in OSRdyTbl[4] is 1.

Bit 5 in OSRdyGrp is 1 when any bit in OSRdyTbl[5] is 1.

Bit 6 in OSRdyGrp is 1 when any bit in OSRdyTbl[6] is 1.

Bit 7 in OSRdyGrp is 1 when any bit in OSRdyTbl[7] is 1.

根据优先级找到任务在就绪任务表中的位置



根据优先级确定就绪表

- 假设优先级为12的任务进入就绪状态，12=1 100b,则OSRdyTbl[1]的第4位置1，且OSRdyGrp的第1位置1，相应的数学表达式为:

```
OSRdyGrp      |=0x02;
OSRdyTbl[1]   |=0x10;
```

- 而优先级为21的任务就绪21=10 101b，则OSRdyTbl[2]的第5位置1，且OSRdyGrp的第2位置1,相应的数学表达式为:

```
OSRdyGrp      |=0x04;
OSRdyTbl[2]   |=0x20;
```

根据优先级确定就绪表

- 从上面的计算我们可以得到:若第n位置1，则应该与2^n 相或。uC/OS中，把2^n的n=0-7的8个值 先计算好存在数组OSMapTbl[7]中,也就是:

OSMapTbl[0] =2^0=0x1;

OSMapTbl[1] =2^1=0x2;

……

OSMapTbl[7] =2^7=0x80;

根据优先级确定就绪表

- 利用OSMapTbl，通过任务的识别号-优先级prio来设置任务在就绪组和就绪表数组中相应位置的数学式为：

```
OSRdyGrp               |=OSMapTbl[prio>>3];
OSRdyTbl[prio>>3]  |=OSMapTbl[prio & 0x07];
```

假设优先级为12，1 100b

```
OSRdyGrp      |=0x02;
OSRdyTbl[1]   |=0x10;
```

根据就绪表确定最高优先级（1）

两个关键:

- 优先级数分解为高三位和低三位分别确定；
- 高优先级有**着小的优先级号**；