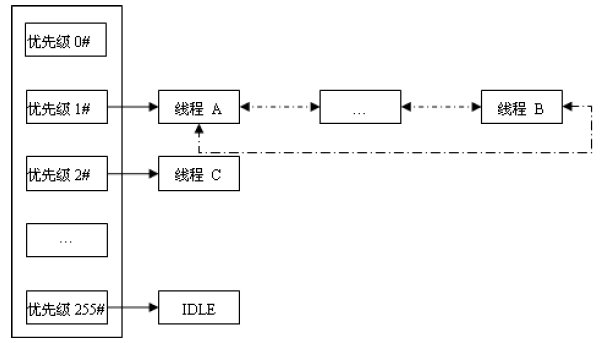
在我看的两种RTOS中，线程都是以优先级队列的方式存储，有的可能支持同优先级的线程，那每一个优先级的线程就以循环链表的方式存储。而这个优先级队列是以数组的方式存储。如下图所示：



在内核调度时，需要从就绪队列中找出优先级最高的线程，乍看下狠高端，其实简化下，就是在数组中找最大数的算法。而且，数的范围是事先知道的。则问题描述如下：

1. 一个固定长度的数组，如32个数的数组。
2. 每个位置代表一个优先级，即数。
3. 每个优先级上可能为空，即没有线程在这个优先级，没有的不参与运算。
4. 在不为空的优先级上找出最小的优先级（即最高的优先级）。

解决这个问题有两个方法。

1. 采用传统方法，遍历就绪队列，找出最小值，但这个时间是不确定的，时间复杂度为O(n)。
2. 采用空间换时间的方法。因为在嵌入式OS中，线程的数目不会很大，通常在256个以内，所以可以事先把所有情况列出。比如系统最多8个线程，假设现在有2，3两个线程，则为 0b00000110，则这种情况下最高优先级为2。那个二进制也是节省空间的做法。

那由上述的第二种做法，假如有8个优先级，我们事先做好优先级表：

01.const unsigned char bitmap[] =

02.{

03. /\* 00 \*/ 0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

04. /\* 10 \*/ 4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

05. /\* 20 \*/ 5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

06. /\* 30 \*/ 4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

07. /\* 40 \*/ 6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

08. /\* 50 \*/ 4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

09. /\* 60 \*/ 5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

10. /\* 70 \*/ 4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

11. /\* 80 \*/ 7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

12. /\* 90 \*/ 4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

13. /\* A0 \*/ 5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

14. /\* B0 \*/ 4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

15. /\* C0 \*/ 6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

16. /\* D0 \*/ 4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

17. /\* E0 \*/ 5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,

18. /\* F0 \*/ 4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0

19.};

这样，任意的线程组合下，都能在O(1)时间内把最高优先级找出来。

当然，还有问题，在8个优先级下，表的体积是可以忍受的，但在32，256这些优先级下，表的体积过大。解决方法是，例如32个优先级，则把32看作是4个字节，针对每个字节采用上述的8优先级表，分别找出每个字节的最值，最后合并，找出最高优先级。