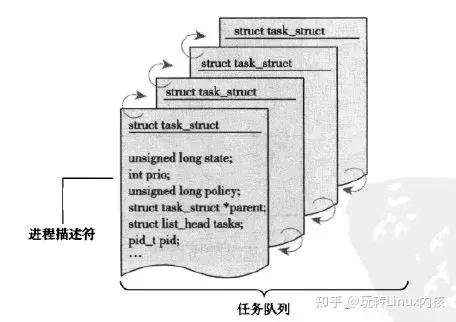
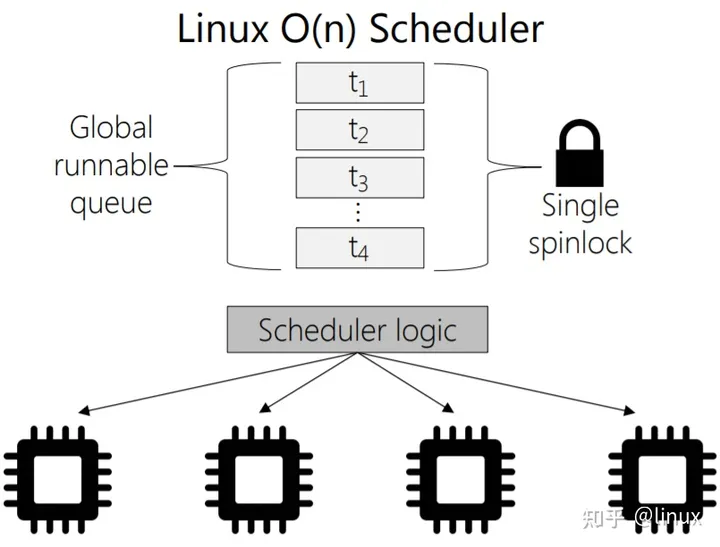
最开始的调度器是O(n)

Linux把所有的进程使用双向链表连接起来, 如下图(来源<Linux设计与实现>):





O(n)调度器采用一个全局队列runqueue作为核心数据结构，具备以下特点：

* 多个cpu共享全局队列，并非每个cpu有单独的队列
* 实时进程和普通进程混合且无序存放，寻找最合适进程需要遍历
* 就绪进程将被添加到队列，运行结束被删除
* 全局队列增删进程任务时需要加锁
* 进程被挂到不同CPU运行

CPU进程队列，采用链表或数组形式，使用时遍历，内部放有带优先级的进程。

缺点，如果队列太长，遍历效率低。

O(n)调度器对实时进程和普通进程采用不同的调度策略：

实时进程采用的是SCHED\_RR或者SCHED\_FIFO，高级优先&同级轮转或者顺序执行

普通进程采用的是SCHED\_OTHER

进程采用的策略在task\_struct中policy体现

搜索下一个合适的进程是基于动态优先级来实现的，动态优先级最高的就是下一个被执行的进程。

O(n)调度器设计和实现上存在一些问题，但是其中的很多思想为后续调度器设计指明了方向，意义深远。

----------------------------

在Linux中进程使用task\_struct结构体来表示。进程描述符的state字段用于保存进程的当前状态, 进程的状态有以下几种:

TASK\_RUNNING (运行) – 进程处于可执行状态, 在这个状态下的进程要么正在被CPU执行, 要么在等待执行(CPU被其他进程占用的情况下).

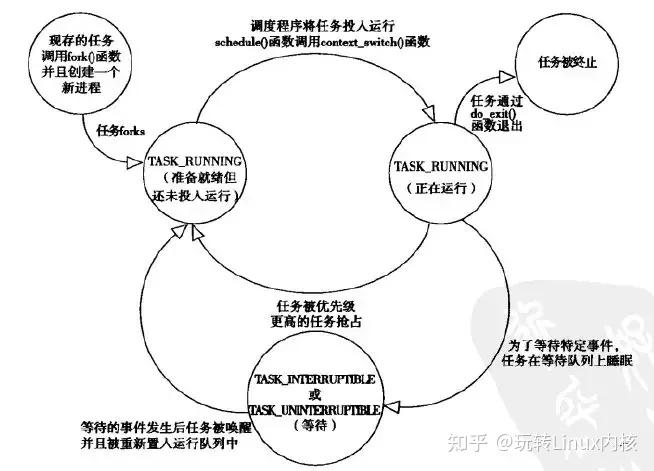
TASK\_INTERRUPTIBLE (可中断等待) – 进程处于等待状态, 其在等待某些条件成立或者接收到某些信号, 进程会被唤醒变为运行状态.

TASK\_UNINTERRUPTIBLE (不可中断等待) – 进程处于等待状态, 其在等待某些条件成立, 进程会被唤醒变为运行状态, 但不能被信号唤醒.

TASK\_TRACED (被追踪) – 进程处于被追踪状态, 例如通过ptrace命令对进程进行调试.

TASK\_STOPPED (停止) – 进程处于停止状态, 进程不能被执行. 一般接收到SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU信号进程会变成TASK\_STOPPED状态.

状态间的转换如下图:



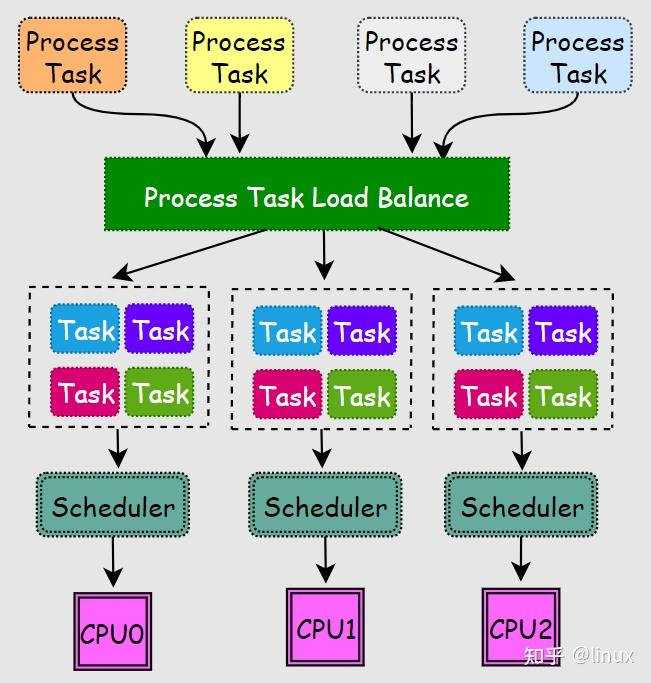
新的算法O(1)调度器

O(n)调度器在linux内核中大约使用了4年，在Linux 2.6.0采纳了Red Hat公司Ingo Molnar设计的O(1)调度算法，该调度算法的核心思想基于Corbato等人提出的多级反馈队列算法。

O(1)调度器引入了多个队列，并且增加了负载均衡机制，对新出现的进行任务分配到合适的cpu-runqueue中（runqueue是一个数据结构，内核为每一颗CPU分配了一个runqueue，用于维护这颗CPU可以运行的进程。<https://zhuanlan.zhihu.com/p/283276298>）：

为了实现O(1)复杂度的pick-next算法，内核实现代码量增加了一倍多，其有以下几个特点：

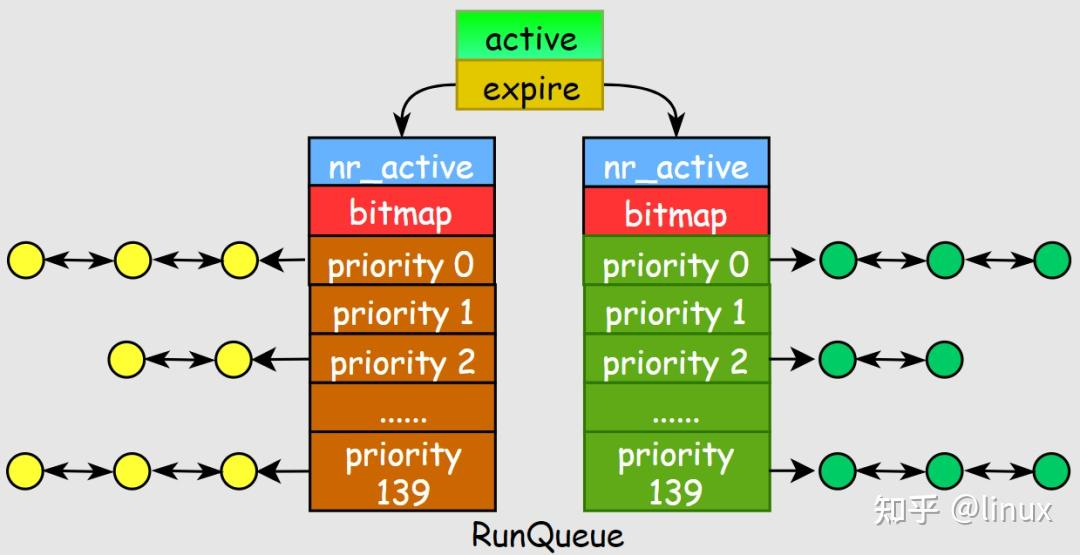
* 实现了per-cpu-runqueue，每个CPU都有一个就绪进程任务队列
* 引入活跃数组active和过期数组expire，分别存储就绪进程和结束进程
* 采用全局优先级：实时进程0-99，普通进程100-139，数值越低优先级越高，更容易被调度
* 每个优先级对应一个链表，引入bitmap数组来记录140个链表中的活跃任务情况

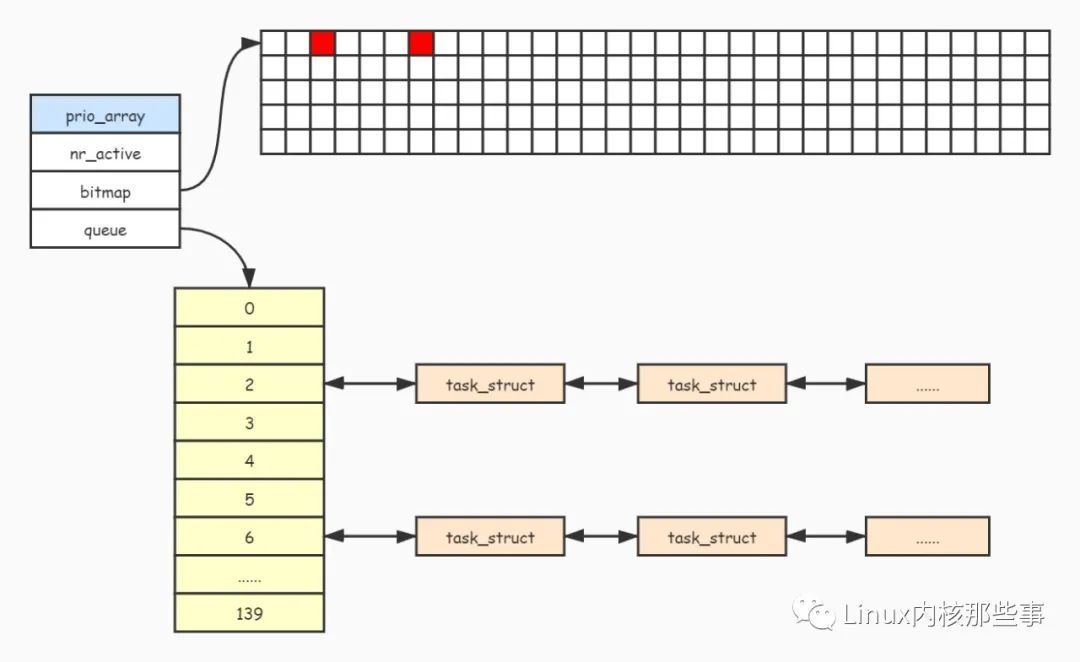


由于计算机内位运算最快，进程优先级共分140级，实时进程0-99，普通进程100-139.用户可调整普通进程参数范围-20-19.

O(1)将优先级映射成bitmap，在大数据中常用于快速排序。

当某个优先级上有进程存在，就把这个位置位为1，这样CPU寻找优先级时只要作位运算，就可以方便的快速找到





CFS调度器

时间片，每个进程相对公平，保证都能分到。

引入virtualtime ,优先级高的进程执行同样实际时间，其virtualtime时间更少。

用红黑树的数组结构来存放virtualtime。

Cpu调度的时候总是去最左边virtualtime最小的进程进行调度。