## 调度策略：

1. 概念：决定什么时候以怎样的方式选择一个新进程运行的这组规则就是所谓的调度策略。
2. Linux的调度基于分时技术：多个进程以“时间多路复用”的方式运行。因为CPU的时间被分成“片”，给每个可运行进程分配一片。当当然，单处理器在任何给定的时刻只能运行一个进程。如果当前运行进程的时间片或时限到期时，该进程还没有运行完毕，进程切换就可以发生。分时依赖于定时中断，因此对进程是透明的。不需要在程序中插入额外的代码来保证CPU分时。
3. 调度策略也是根据进程的优先级对它们进行分类。在Linux中，进程的优先级是动态的。调度程序跟踪进程正在做什么，并周期性地调整它们的优先级。这种方式下，在较长的时间间隔内没有使用CPU的进程，通过动态地增加它们的优先级来提升它们。相应的，对于已经在CPU上运行了较长时间的进程，通过减少它们的优先级来处罚它们。
4. 与调度相关的系统调用  
   nice()：改变一个普通进程的静态优先级

getpriority()：获得一组普通进程的最大静态优先级

setpriority()：设置一组普通进程的静态优先级

sched\_getcheduler()：获得一个进程的调度策略

sched\_setcheduler()：设置一个进程的调度策略和实时优先级

sched\_getparam()：获得一个进程的实时优先级

sched\_setparam()：设置一个进程的实时优先级

sched\_yield()：自愿放弃处理器而不阻塞

sched\_get\_priority\_min()：获得一种策略的最小实时优先级

sched\_get\_priority\_max()：获得一种策略的最大实时优先级

sched\_rr\_get\_interval()：获得时间片轮转策略的时间片值

sched\_setaffinity()：设置进程的CPU亲和力掩码

sched\_getaffinity()：获得进程的CPU亲和力掩码

1. Linux的进程是抢占式的，无论进程处于内核态还是用户态，都可能被抢占。如果一个进程进入TASK\_RUNNING状态，内核检查它的动态优先级是否大于当前正在运行的进程的优先级。如果是，current的执行被中断，并调用调度程序选择另一个进程运行（通常是刚刚变为可运行的进程）。当然，进程在它的时间片到期时也可以被抢占。此时，当前进程的thread\_info结构中的TIF\_NEED\_RESCHED标志被设置，以便中断处理程序终止时调度程序被调用。

## 时间片长度：

1. 如果时间片太短，有进程切换引起的开销就变得非常高。如果时间片太长，进程看起来就不再是并发执行的。通常认为时间片长会降低交互式应用程序的响应时间，这是错误的，交互式进程相对有较高的优先级，因此，不管时间片多长，它们都会很快的抢占批处理进程。
2. Linux选择尽可能长、同时能保持良好响应时间的一个时间片。

## 调度算法：

1. Linux中，进程被调度时的类型

（1）SCHED\_FIFO：

先进先出的实时进程。当调度程序把CPU分配给进程的时候，它把该进程描述符保留在运行队列链表的当前位置。如果没有其他可运行的更高优先级的实时进程，进程就继续使用CPU，想用多久就用多久，即使还有其他具有相同优先级的实时进程处于可运行状

（2）SCHED\_RR：

时间片轮转的实时进程。当调度程序把CPU分配给进程的时候，它把该进程的描述符放在运行队列链表的末尾。这种策略保证对所有具有相同优先级的实时进程公平地分配CPU时间。

（3）SCHED\_NORMAL：

普通的分时进程

// 调度算法根据进程是实时进程还是普通进程有很大的不同。

1. 普通进程的调度：
2. 每个普通进程都有它自己的静态优先级，内核用从100（最高静态优先级）到139（最低静态优先级）的数表示普通进程的静态优先级。值越大静态优先级月底。
3. 新进程总是继承其父进程的静态优先级。不过，通过把某些“nice值”传递给系统调用nice()和setpriority()，用户可以改变自己拥有的进程的静态优先级。
4. 静态优先级本质上决定了系统分配给进程的时间片长度。

公式：基本时间片 = (140 - 静态优先级) \* 20，静态优先级<120

= (140 – 静态优先级)\*5，静态优先级>=120

1. 普通进程除了拥有静态优先级，还有动态优先级，其值的范围是100（最高静态优先级）到139（最低静态优先级）。动态优先级是调度程序在选择新进程来运行的时候使用的数。它与静态优先级的关系用下面的公式：

动态优先级 = max(100， min(静态优先级 - bonus + 5, 139));

// bonus是范围从0~10的值，值小于5表示降低动态优先级以示惩罚，值大于5表示增加动态优先级以示奖赏。bonus的值依赖于进程过去的情况，是与进程的平均睡眠时间有关。平均睡眠时间是进程在睡眠状态所消耗的平均纳秒数。

1. 活动进程：进程没有用完它们的时间片，因此允许它们运行

过期进程：这些可运行进程已经用完了它们的时间片，因此被禁止运行，直到所有活动进程都过期。

3、实时进程的调度：

（1）每一个进程都与一个实时优先级相关，实时优先级是一个范围从1（最高优先级）~99（最低优先级）的值。调度程序总是让优先级高的进程运行，实时进程运行的过程中，禁止低优先级的进程执行。与普通进程相反，实时进程总是被当作活动进程。用户可以调用sched\_setparam()和sched\_setcheduler()改变进程的实时优先级。

（2）如果几个可运行的实时进程具有相同的最高优先级，那么调度程序选择第一个出现在本地CPU的运行队列相应链表中的进程。

（3）只有下述事件之一发生时，实时进程才会被另外一个实时进程取代：

A、进程被另外一个具有更高实时优先级的实时进程抢占。

B、进程执行了阻塞进程并进入睡眠（处于TASK\_INTERRUPTIBLE或TASL\_UNINTERRUPTIBLE状态）

C、进程停止（处于TASK\_STOPPED或TASK\_TRACED状态）或被杀死（处于EXIT\_ZOMBIE或EXIT\_DEAD状态）

D、进程通过调用系统调用sched\_yield()，自愿放弃CPU

E、进程是基于时间片轮转的实时进程，而且用完了它的时间片

（4）当系统调用nice()和setpriority()用于基于时间片轮转的实时进程时，不改变实时进程的优先级而会改变基本时间片的长度。实际上，基于时间片轮转的实时进程的基本时间片的长度与实时进程的优先级无关，而依赖于进程的静态优先级（时间片长度根据静态优先级计算出来）。

## 调度程序所使用的数据结构：

1. 数据结构runqueue：
2. 系统中每个CPU都有它自己的运行队列，所有runqueue结构存放在runqueue每CPU变量中。宏this\_rq()产生本地CPU运行队列的地址，而宏cpu\_rq(n)产生索引为n的CPU的运行队列的地址。
3. 字段：

类型 名称 说明

spinlock\_t lock 保护进程链表的自旋锁

unsigned long nr\_running 运行队列中可运行进程的数量

unsigned long cpu\_load 基于运行队列中进程的平均数量的CPU负载因子

unsigned long nr\_switches CPU执行进程切换的次数

unsigned long nr\_uninterruptible 先前在运行队列链表中而现在睡眠，在TASK\_UNINTERRUPTIBLE状态的进程的数量

unsigned long expired\_timestamp 过期队列中最老的进程被插入队列的时间

unsigned long long timestamp\_last\_tick 最近一次定时器中断的时间撮的值

task\_t \* curr 当前正在运行进程的进程描述符指针

task\_t \* idle 当前CPU上swapper进程的进程描述符指针

struct mm\_struct\* prev\_mm 在进程切换时用来存放被替换进程的内存描述符的地址

prio\_array\_t\* active 指向活动进程链表的指针

prio\_array\_t\* expired 指向过期进程链表的指针

prio\_array\_t[2] arrays 活动进程和过期进程的两个集合

int best\_expired\_prio 过期进程中静态优先级最高的进程

atomic\_t nr\_iowait 先前在运行队列的链表中而现在正在等待磁盘I/O操作结束的进程的数量

struct sched\_domain\* sd 指向当前CPU的基本调度域

int active\_balance 如果要把一些进程从本地运行队列迁移到平衡队列，就设置这个标志

int push\_cpu 未使用

task\_t\* migration\_thread 迁移内核线程的进程描述符指针

struct list\_head migration\_queue 从运行队列中被删除的进程的链表

1. 系统中的每个可运行进程属于且只属于一个运行队列。只要可运行进程保持在同一个运行队列中，它就只可能在拥有该运行队列的CPU上执行。
2. 运行队列的arrays字段是一个保安两个prio\_array\_t结构的数组。每个数组都表示一个可运行进程的集合，并包括140个双向链表表头（每一个链表对应一个可能的进程优先级）、一个优先级位图和一个集合中所包含的进程数量的计数器。

## 调度程序所使用的函数：

1. 重要的函数：

（1）scheduler\_tick() 维持当前最新的time\_slice计数器。每次时钟节拍到来时，scheduler\_tick()被调用以执行与调度有关的操作。为了保持系统中运行队列的平衡，每次经过一次时钟节拍，scheduler\_tick()就调用reblance\_tick()函数。

（2）try\_to\_wake\_up() 唤醒睡眠进程。把进程状态设置为TASK\_RUNNING，并把进程插入到本地CPU的运行队列来唤醒睡眠。

（3）recalc\_task\_prio() 更新进程的动态优先级和平均睡眠时间。

（4）schedule()

A、选择要执行的新进程。从运行队列的链表中找到一个进程，并随后将CPU分配给这个进程。延时调用方式：由于总是在恢复用户态进程的执行之前检查TIF\_NEED\_RESCHED标志，所以这个标志为1，就会调用schedule()函数。

B、该函数的关键结果是设置一个叫做next的变量，使它指向被选中的进程，该进程将取代current进程。如果系统中没有优先级高于current进程的可运行进程，那么最终nenxt与current相等，不发生任何进程切换。

（5）load\_balance() 维持多处理器系统中运行队列的平衡

（6）nice()系统调用允许进程改变他们的基本优先级。

（7）sched\_getParam()：为参数Pid所表示的进程检索调度参数。

## 调度域：

1. 概念：调度域实际上是一个CPU集合，它们的工作量应当由内核保持平衡。一般来说，调度域采取分层的组织形式：最上层的调度域（通常包括系统中的所有CPU）包括多个子调度域，每个子调度域包括一个CPU子集。
2. 每个调度域被依次划分为一个或多个组，每个组代表调度域的一个CPU子集。工作量的平衡总是在调度域的组之间来完成的。换言之，只有在某个调度域的某个组的总工作量远远低于同一个调度域的另外一个组的工作量时，才把进程从一个CPU迁移到另一个CPU。
3. 每个调度域由一个sched\_domain描述符表示，而调度域中的每个组由sched\_group描述符表示。每个sched\_domain描述符包括一个groups字段，它指向组描述符链表中的第一个元素。此外，sched\_domain结构的parent字段指向父调度域的描述符（如果有的话）。
4. 系统中所有物理CPU的sched\_domain描述符都存放在每CPU变量phys\_domains中。如果内核不支持超线程技术，这些域就在域层次结构的最底层，运行队列描述符的sd字段指向它们，即它们是最基本调度域。相反，如果内核支持超线程技术，则底层调度域存放在每CPU变量cpu\_domains中。