最基础的处理器调度（简化的处理器调度）：

Round-Robin（RR）

时间片轮转调度算法，每隔一定时间进行中断，在处理中断的过程中，试图去切换到下一个线程或进程，如果下一个线程或进程正在等待I/O操作，则继续尝试下一个，如果所有线程都不需要CPU，则调度idle线程执行。

中断之间的进程或线程执行称为“时间片”。

这个调度算法非常地简单但不够优秀，例如：一个vim加上多个其他空程序。

改进策略：**引入进程的优先级**。然后我们的调度算法要基于优先级进行调度。

UNIX：niceness （-20 - 19）越nice越让别人得到CPU。

基于niceness的调度方法：RTOS：坏人躺下，好人才能站起来。（在某些情况下很有用，但是桌面系统不好）

Linux的方案：nice相差10，CPU资源获得率相差10倍。

真实的处理器调度：

RR的问题：系统有两个进程：1.交互式的Vim、单线程，2.纯粹计算的线程。

RR调度可能会使得Vim的响应时间很长，因为Vim的时间片被纯粹计算的线程占用了。

一个想法：交互式的程序不会是CPU密集型的。

策略：

动态优先级（MLFQ）

设置若干个RR队列，每个队列对应一个优先级。

动态优先级策略：

- 优先调度高优先级队列
- 用完时间片：降低优先级（坏人 ---> 好人）
- 主动让出CPU（I/O密集型）：提高优先级（好人 ---> 坏人）

但还是有问题：例如：知道调度算法后可以hack调度算法。

```
while(1){
   if(/*如果时间片快用完了*/){
      sleep(/*一定时间*/);
   }
}
```

而且并不是所有的I/O密集型CPU使用率都不高。

需要定期把优先级拉平

完全公平调度（CFS）

今天正在使用的Linux的调度。

“让系统里的所有进程尽可能公平地共享处理器”

每个进程都有一个时钟（取决于进程的优先级），然后每次调度时，选择时钟最小的进程执行，执行相同时间（虚拟时间）。

统计时间时除以**sched_prio_to_weight**.

CFS 的复杂性（1）：新进程/线程

调度器要决定，fork完成之后，是否要让子进程执行。

以前的Linux是child first。但现在不是了。

从2.6.32开始，Linux是parent first。

问题是，fork出来的新进程应该赋多少virtual runtime？以前会让子进程多跑一点，但是多fork的代码表现不好，现在是直接继承父进程的virtual runtime。

CFS 的复杂性（2）：I/O

某个进程sleep了1s，导致其virtual runtime非常落后，那么根据以前的调度策略，这个进程接下来会独占cpu，因为要补齐virtual runtime。

这要怎么做呢？需要设计。

CFS 的复杂性（3）：整数溢出

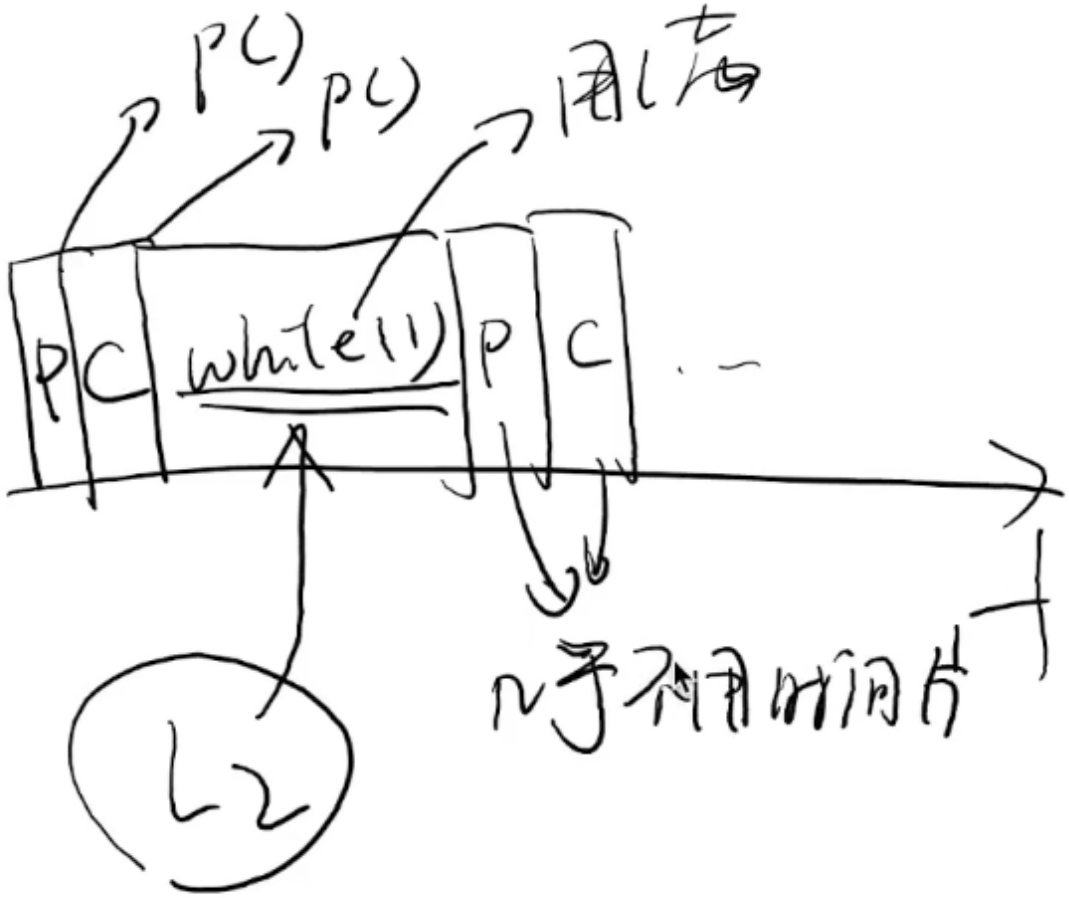
假设：系统中最近、最远的时刻相差不超过数轴的一半，我们比较他们的相对大小。

CFS 的实现：

需要用一个有序的集合来维护，内核用的红黑树。

考虑一个更复杂的情况：

当系统中出现了生产者-消费者情况，还有一个while1的循环，时间片的使用如图所示：



那么在三种算法中的表现是这样的：

- RR: producer-costumer会取得时间非常少的cpu。
- MLFQ: producer-costumer会在高优先级队列中，对CPU密集型（while1）不太公平。
- CFS有比较好的表现。

另外一个问题：

在持有互斥锁的时候被赶出去，校长因为需要锁，所以也会等待，所以，最后是系主任持有cpu在等待。我们称之为：“优先级翻转”。

解决优先级翻转：

- 优先级继承/优先级提升：低优先级的进程block高优先级的进程时，把低优先级的进程的优先级提高到高优先级。

但是对于条件变量还是没用。

但是真正地问题是：**多处理器调度**

多处理器调度的困难所在

既不能简单地“分配线程到处理器”

- 线程退出，瞬间处理器开始围观

也不能简单地“谁空丢给谁”

- 在处理器之间迁移会导致 cache/TLB 全都白给

多处理器调度的两难境地

- 迁移？可能过一会儿还得移回来
- 不迁移？造成处理器的浪费

比如说在多用户时，可以在依据进程组进行调度。