可以证明，对于一批同时到达的任务，采用 SJF 算法将得到一个最小的平均周转时间。例如，假设有四个任务 A、B、C、D，它们的运行时间分别是 a、b、c 和 d，假设它们的到达时间是差不多的，调度顺序为 A、B、C、D。那么任务 A 的周转时间为 a，B 的周转时间为 a+b，C 的周转时间为 a+b+c，D 的周转时间为 a+b+c+d，因此，最后的平均周转时间为(4a+3b+2c+d)/4，从这个式子来看，显然，只有当 a < b < c < d 的时候，这个平均周转时间才会达到一个最小值。这个结论可以推广到任意多个任务的情形。

3）时间片轮转算法

时间片轮转算法（Round Robin，RR）的基本思路是：把系统当中的所有就绪任务按照先来先服务的原则，排成一个队列。然后，在每次调度的时候，把处理器分派给队列当中的第一个任务，让它去执行一小段 CPU 时间，或者叫时间片。当这个时间片结束的时候，如果任务还没有执行完的话，将会发生时钟中断，在时钟中断里面，调度器将会暂停当前任务的执行，并把它送到就绪队列的末尾，然后执行当前的队首任务。反之，如果一个任务在它的时间片用完之前就已经运行结束了或者是被阻塞了，那么它就会立即让出 CPU 给其他的任务。

时间片轮转法的优点是：

- 公平性：各个就绪任务平均地分配 CPU 的使用时间。例如，假设有 n 个就绪任务，那么每个任务将得到 $1/n$ 的 CPU 时间。
- 活动性：每个就绪任务都能一直保持着活动性，假设时间片的大小为 q，那么每个任务最多等待$(n-1)q$ 这么长的时间，就能再次得到 CPU 去运行。

在采用时间片轮转算法时，时间片的大小 q 要适当选择，如果选择不当，将影响到系统的性能和效率。

- 如果 q 太大，每个任务都在一个时间片内完成，这就失去了轮转法的意义，退化为先来先服务算法了，这就使各个任务的响应时间变长。
- 如果 q 太小，每个任务就需要更多的时间片才能运行完，这就使任务之间的切换次数增加，从而增大了系统的管理开销，降低了 CPU 的使用效率。

因此，如何来选择一个合适的 q 值，既不能太大，也不能太小，这是时间片轮转法的最大问题。一般来说，这个值选在 20～50ms 是比较合适的。

4）优先级算法

优先级调度算法的基本思路是：给每一个任务都设置一个优先级，然后在任务调度的时候，在所有处于就绪状态的任务中选择优先级最高的那个任务去运行。例如，短作业优先算法其实也是一个优先级算法，每个任务的优先级就是它的运行时间，运行时间越短，优先级越高。

优先级算法可以分为两种：可抢占方式和不可抢占方式。它们的区别在于：当一个任务正在运行的时候，如果这时来了一个新的任务，其优先级更高，那么在这种情形下，是立即抢占 CPU 去运行新任务，还是等当前任务运行完了再说。