#### 典型回答

CPU 调度是多道程序操作系统的基础。通过在进程间切换 CPU，操作系统可以使得计算机更加高效。每当 CPU 空闲时，操作系统就应从就绪队列中选择一个进程来执行。CPU 调度算法有以下几种：

1. 先到先服务调度
2. 最短作业优先调度
3. 优先级调度
4. 轮转调度
5. 多级队列调度
6. 多级反馈队列调度

先来先服务（FCFS）调度是最简单的调度算法，但是它会让短进程等待很长的进程。最短作业优先调度（SJF）调度可证明是最佳的，提供最短的平均等待时间，然而，SJF 的实现是很难的，因为预测下一个 CPU 执行的长度是难的。SJF 算法是通用优先级调度算法（简单分配 CPU 到具有最高优先级的进程）的一个特例。优先级和 SJF 的调度可能产生饥饿。老化技术阻止饥饿。

轮转（RR）调度更适合于分时（交互）系统。RR 调度为就绪队列的首个进程，分配 q 个时间单位，这里 q 是时间片。在 q 个时间单位之后，如果该进程还没有释放 CPU，那么它被抢占并添加到就绪队列的尾部。该算法的主要问题是选择时间片，如果时间片太大，那么 RR 调度就成了 FCFS 调度；如果时间片太小，那么由于上下文切换引起的调度开销就过大。

FCFS 调度算法是非抢占的，而 RR 调度算法是抢占的。SJF 和优先级算法可以是抢占的，也可以是非抢占的。

多级队列算法允许多个不同算法用于不同类型的进程。最常用的模型包括：使用 RR 调度的前台交互队列与使用 FCFS 调度的后台批处理队列。多级反馈队列允许进程在队列之间迁移。

#### 知识延伸

对于支持线程的操作系统，操作系统实际调度的是内核级线程而非进程。

用户级线程是由线程库来管理的，而内核并不知道它们。用户级线程为了运行在 CPU 上，最终应映射到相关的内核级线程，但是这种映射可能不是直接的，可能采用轻量级进程（LWP）。

**多处理器调度**

对于多处理器系统，CPU 调度的一种方法是让一个处理器处理所有调度决定、I/O 处理以及其他系统活动，其他处理器只执行用户代码。这种非对称多处理很简单，因为只有一个处理器访问系统数据结构，减少了数据共享的需要。

第二种方法是使用对称多处理（SMP），即每个处理器自我调度。所有进程可能处于一个共同的就绪队列中，或每个处理器都有它自己的私有就绪进程队列。不管如何，调度这样进行：每个处理器的调度程序都检查共同就绪队列，以便选择执行一个进程。

大多数 SMP 系统试图避免将进程从一个处理器移到另一个处理器，而是试图让一个进程运行在同一个处理器上。这称为处理器的亲和性，即一个进程对它运行的处理器具有亲和性。

对于 SMP 系统，最重要的是保持所有处理器的负载平衡，以便充分利用多处理器的优点。负载平衡设法将负载平均分配到 SMP 系统的所有处理器。负载平衡通常有两种方法：推迁移和拉迁移。但是，负载平衡往往会抵消掉处理器亲和性的好处，也就是说，保持进程运行在同一个处理器上的好处是进程可以利用它在该处理器缓存内的数据。

接下来，我们讨论 Linux 和 Windows 的调度策略。重要的是要注意，我们使用的进程调度这一术语是泛指的。事实上，在讨论 Windows 系统时，采用内核线程调度，而在讨论 Linux 时，采用任务调度。

**Linux 调度**

完全公平调度程序（CFS）是默认的 Linux 调度算法。

Linux 系统的调度基于调度类，每个类都有一个特定的优先级。内核针对不同的调度类，采用不同的调度算法，以便满足系统与进程的需要。Linux 标准内核实现了两个调度类：采用 CFS 调度算法的默认调度类和实时调度类。

CFS 调度程序并不采用严格规则来为一个优先级分配某个长度的时间片，而是为每个任务分配一定比例的 CPU 处理时间。

Linux CFS 调度程序采用高效算法，以便选择运行下个任务。每个可运行的任务放置在红黑树上。

当一个任务变得可运行时，它被添加到树上。当一个任务变得不可运行时，它从树上删除。一般来说，得到较少处理时间的任务会偏向树的左侧；得到较多处理时间的任务会偏向树的右侧。由于红黑树是平衡的，找到最左侧结点需要 lg N 的·时间复杂度。从 CFS 调度程序角度而言，这也是具有最高优先级的任务。

**Windows 调度**

Windows 采用基于优先级的、抢占调度算法来调度线程。Windows 调度程序确保具有最高优先级的线程总是在运行的。用于处理调度的 Windows 内核部分称为调度程序。

调度程序采用 32 级的优先级方案，以便确定线程执行方案。