进程调度

调度:有一堆任务要处理,由于资源有限,这此事情没法同时处理。就需要确定某种规则来决定处理这些任务的顺序。在多道程序中，进程的数量往往是多于处理机个数的。这样不可能同时并行的处理各个进程。处理机调度，就是从就绪队列中按一定的算法选一个进程并将处理机分配给它运行，以实现进程的并发执行。

1.调度的三个层次

1.高级调度

由于内存空间有限，无法将用产提交的作业全部到内存,要规定某种规则来决定作业调入内存的顺序。

首先，我们要从外部后备队列中挑选作业，并为它们分配必要的内存资源，同时创建相应的进程控制块（PCB），以便它们可以获得对处理机的访问权限。高级别调度主要涉及到作业的调度，即将作业调入内存时会建立相应的PCB，而作业调出时则会释放PCB。高级调度主要是在内存和外部存储之间的调度，每个作业只能被调入一次，作业调出则是在作业执行结束后进行。

2.中级调度

中级调度（也称为内存调度）是一种稳定状态，它可以将那些暂时无法在内存中运行的进程挂起，并将它们重新定位到外部存储器。当重新满足一定条件并且有足够的内存空间时，再将它们调回内存中。中级调度的主要目的是尽可能地利用系统资源，并确保在内存中的进程数可以在给定的时间段内保持在一个合理的范围内。操作系统会记录进程数据在外部存储器中的存储位置，同时通过内存中的进程控制块（PCB）来管理每个进程的状态和信息。被挂起的进程在重新调入内存时，其对应的PCB仍然保留在内存中。

3.低级调度

低级调度（也称为进程调度）主要目的是根据某种策略从就绪队列中选择一个进程以执行。低级调度是指系统中最基本的调度类型，在一般的操作系统中都必须配备进程调度功能。进程的调度频率很高，通常是以毫秒为单位。

2.进程调度的时机

之前提到的高级、中级和低级调度都是根据某种规则或策略来选择调度作业或进程的过程。

当前系统运行时，需要进行进程调度以处理进程的切换情况，例如：

- 当进程正常终止运行或发生异常而终止时，需要主动请求阻塞进程，如等待I/O操作。
- 另外，系统也会给予进程一定的时间片，如果时间片用尽，进程也会被调度。
- 进程可能会被动放弃处理器，让出执行权给有更高优先级的进程，这通常发生在高优先级进程进入就绪队列时。
- 系统还允许进程被动放弃处理器，以处理更紧急的任务，比如硬件中断（如强制剥夺进程占用处理器的权力）。

在某些情况下，无法进行进程调度和切换，例如：

- 在中断处理程序中，处理中断的过程复杂且与硬件密切相关，很难在处理中断后立即进行进程切换。
- 在操作系统的临界区中，通常是在原子操作中进行。在临界区中，进程无法被调度和切换，但在一般的临界区内允许调度和切换。

3.进程调度的方式

非剥夺式调度方式，又称非抢占方式，即只允许进程主动放弃处理器。在运行过程中，即使有更紧急的任务到来，当前进程依然会继续使用处理器，直到该进程终止或主动要求进入阻塞态。这种方式实现简单，系统开销小，但无法立即处理紧急任务，通常用于批处理系统。

与之相对的是剥夺式调度方式，也称为抢占方式。当一个进程在处理器上运行时，如果有更重要、更紧急的进程需要使用处理器，则会暂停正在执行的进程，将处理器分配给更重要、紧急的进程。这种方式常用于分时和实时系统。

狭义的进程调度与进程切换的区别在于：

- 狭义的进程调度是指从就绪队列中选择一个进程进行调度，这个进程可能是刚刚被唤醒的进程，也可能是另一个进程。而后者则需要进行进程切换。
- 进程切换是指一个进程正被处理机使用，然后由另一个进程来担任处理机的过程。进程调度是选择一个进程的过程，而进程切换则是完成了选择之后的操作。

进程切换的步骤包括：

1. 对原来运行进程的各种数据进行保存，包括程序运行结果、地址等，存放到相应的数据结构中，例如 PCB（进程控制块）。
2. 对新的进程的各种数据进行加载，包括从内存中读取数据，存放到寄存器中，准备开始执行新进程的运行。这些信息通常包含在 PCB 中。

需要注意的是，进程切换是有代价的。如果频繁进行进程切换，会使整个系统的效率降低，因为系统大部分时间都花在了进程切换上，而不是用于执行进程的实际工作。

4.调度算法的评价指标

周转时间是指从作业进入外存后备队列开始，到作业完成为止的选段时间间隔，可分为四个部分：作业在外存后备队列上等待作业调度（高级调度）的时间、进程在就绪队列上等待进程调度（低级调度）的时间、进程在CPU上执行的时间（运行态）、进程因I/O操作而阻塞的时间（阻塞态）。

作业周转时间包括作业完成时间与作业提出变化时间。对用户而言，更关注的是单个作业的周转时间，即从作业进入内存开始到作业在内存中被处理完成为止的时间。

作业的响应时间是指从用户提出请求到系统首次响应所用的时间。进程的响应时间包括进程到达就绪队列的时间和进程第一次获取 CPU 的时刻。对用户而言，响应时间越短越好。

5.调度算法

1.FCFS

1. 先来先服务（FCFS，First Come First Serve）算法思路：主要从“公平”的角度考虑（类似生活中排队购买东西）。
2. 算法规则：按照作业/进程到达的先后顺序进行服务。用于作业调度时考虑谁先到达后备队列；用于进程调度时，谁先到达就绪队列。这是一种非抢占式的算法。
3. 优缺点：
    - 优点：公平、算法实现简单。
    - 缺点：短作业可能需要长时间等待，导致带权周转时间较大，用户体验不佳。对于短作业来说，用户体验不好。FCFS 算法对长作业有利，对短作业不友好。
4. 是否会导致饥饿：由于排队的原因，后面的进程不会被插队，前面的进程会优先得到处理。因此，不会有进程直接在就绪队列等待服务而被忽视。

2.短作业优先

1. 短作业优先（SJF，Shortest Job First）算法思想：追求最少的平均等待时间、最少的平均周转时间、以及最少的平均带权周转时间。
2. 算法规则：最短的作业/进程优先得到服务（“最短”指的是所需处理机服务的时间最短）。适用于作业/进程调度时称为"短作业优先算法"（SJF）。

3. 是否可抢占？SJF是一种非抢占式算法（即不允许动态放弃），但也有抢占式的变种，称为SRTN（Shortest Remaining Time Next）。SRTN要求当有新进程到达时，如果它的执行时间比当前运行进程的剩余执行时间还短，则会抢占处理机，当前运行进程回到就绪队列。此外，当进程完成时主动放弃处理机也需要调度。
4. 优缺点：
    - 优点：最短的等待时间，提高了系统的吞吐量，对于短作业/进程表现较好。
    - 缺点：不公平，短作业可能会拥挤长作业，可能导致饥饿现象。对于只依靠用户提供的下次运行时间的情况，无法确保长作业优先得不到处理的情况。
5. 是否会饥饿：会。如果长时间得不到服务，进程可能会因饥饿而无法执行。

3.高响应比优先

高响应比优先（HRRN，Highest Response Ratio Next）算法思想：综合考虑作业/进程的等待时间和要求服务时间，选择响应比最高的作业/进程进行服务，其中响应比被定义为要求服务时间与等待时间的比值。

1. 算法规则：在每次调度时，计算每个作业/进程的响应比，并选择响应比最高的作业/进程为其服务。要求服务时间相同时，优先选择等待时间越长的作业/进程。
2. 用于作业/进程调度：适用于作业调度和进程调度。
3. 是否可抢占：HRRN是一种非抢占的算法，因此只有当当前运行作业/进程主动放弃处理机时，才需要重新计算响应比。
4. 优缺点：
    - 优点：综合考虑了长等待时间和要求服务时间的情况，使得要求服务时间较短的作业/进程优先得到服务（类似于短作业优先算法的优点）。同时，对于长作业/进程，随着等待时间的增长，其响应比也会增加，从而避免了长作业/进程可能出现的饥饿问题。
    - 缺点：FCFS（先来先服务）/SJF（短作业优先）/HRRN适用于批处理系统，追求的是整体性能指标，而不是关注任务的紧急程度或用户交互性等因素。
5. 是否会出现饥饿：不会。

4.时间片轮转

时间片轮转（RR，Round-Robin）算法思想：公平地轮流为各个进程提供服务，确保每个进程在一定时间间隔内都能得到响应。

1. 算法规则：按照各进程到达就绪队列的顺序，轮流为它们分配时间片。若一个进程未能在一个时间片内完成执行，则将其剩余的执行时间重新放到就绪队列的队尾，以便后续继续执行。
2. 用于作业/进程调度：用于进程调度（只有作业放入内存后才可能需要被分配处理机时间片）。

3. 是否可抢占：可以，若进程未能在时间片内完成执行，则会被强制剥夺处理机的使用权，因此RR属于抢占式算法。
4. 优缺点：
    - 优点：公平、响应快，适用于分时操作系统。时间片合理设置时，能够均衡处理各个进程，确保没有进程长时间得不到执行。
    - 缺点：由于频繁的进程切换会带来一定的开销，不适用于对任务的紧急程度有较高要求的情况，因为可能会导致某些紧急任务长时间得不到处理。
5. 是否会出现饥饿：不会，因为时间片轮转会公平地为进程提供服务，确保每个进程都有机会执行。

5.优先级调度算法

1. 优先级调度算法（优先数越高，优先级越高）。相同时，此算法将处理机分配给优先级较高的作业/进程。
2. 算法思想：随着计算机的发展，特别是实时系统的出现，越来越多的任务应用需要根据任务的紧急程度和重要性来决定处理顺序。
3. 算法规则：每个作业/进程都有自己的优先级，调度时选择优先级最高的作业/进程。
4. 用于作业/进程调度：可用于作业调度和进程调度，甚至会用于I/O事件的处理中（这种情况下通常是非抢占式的，即在进程主动放弃处理机时进行优先级比较）。
5. 是否可抢占：可抢占，当有新作业进入就绪队列时，会进行优先级比较，若新作业的优先级高则抢夺当前进程的处理机。
   PS: 就绪队列未必只有一个，可以按照不同优先级来组织，有的操作系统会根据优先级调整就绪队列，让优先级高的进程在队列中优先执行。

- 静态优先级：创建进程时确定之后一直不变。
- 动态优先级：创建时有个初始值，之后会根据情况动态地调整优先级。
- 固定优先级：系统进程高于用户进程，前台进程高于后台进程。
- I/O型进程（或I/O繁忙型进程）：I/O设备和CPU是可以并行工作的。如果让I/O繁忙型进程优先运行的话，则越有可能让I/O设备尽早投入使用，从而提高系统原始利用率，系统吞吐量都会得到提升。I/O型进程可能调用I/O设备并行执行，这样可以处理更多的任务，提高系统的可处理任务数量。
- 计算型进程（CPU繁忙型进程）：因为需要持续使用CPU，所以什么时候调整优先级以及动态优先级中从追求公平的提高资源利用率的南度考虑了很久。

1. 优缺点：
    - 优点：能够用优先级区分任务的紧急程度、重要程度，适用于实时操作系统，可根据具体情况动态调整对各进程/作业的偏好程度。

- 缺点：若原始顺序不得当或有高优先级进程频繁进入，则可能导致调整/变化的优先级不公平，会出现饥饿现象。
- 是否会出现饥饿：会，若调整/变化的优先级不公平，则可能导致某些进程长时间得不到处理，从而产生饥饿问题。

6.多级反馈队列算法

多级反馈队列算法的级反馈队列的最底层采用FCFS算法。

1. 算法思想：对其他调度算法的折中权衡。
2. 算法规则：设置多个级别的就绪队列，按照优先级从高到低排列，时间片从小到大分配。
3. 进程到达时首先进入最高级别队列，按照FCFS原则等待被分配时间片，若时间片用完进程仍未结束，则进入下一级别的就绪队列队尾，如果此时已经是最底层队列则重新进入队尾。
4. 用于作业/进程调度：用于进程调度。
5. 是否可抢占：是抢占式算法。在较高级别队列的进程执行时，若更高级别队列中有新进程到达，则新进程会进入优先级更高的队列中，因此新进程会抢占处理机，原来运行的进程则放回上一级队列队尾。
6. 优缺点：
   - 对各类型进程相对公平（FCFS的优点）。
   - 每个新到达的进程都能快速得到响应（RR的优点）。
   - 短进程只需要短暂的处理时间（SPF的优点），短进程有可能在前几级队列被及时处理。
   - 可灵活地调整对各类进程的偏好程度。
   - 不必实现估计进程层行时间，避免用户作弊，相比SJF，短进程不会因为长作业的到来而被阻塞。
7. 是否会导致饥饿：会。原本没有进入高优先级队列的短进程可能会一直处于低优先级队列而得不到CPU服务。

多级反馈队列算法是近三种调度算法中最常用的交互式系统调度算法，能够保证响应时间较短，用户体验较好。