一、进程和线程

1.进程的定义

程序的并发执行指的是多个程序都处于已经开始，但都未执行完毕的状态中，在单CPU的计算机系统中，对并发执行的每个程序的执行而言，是一种停停走走地向前推进的模式。在这种情况下，如果仍然使用已有的程序这个概念，只能对它进行静止的、孤立的研究，不能动态地反应并发程序的活动和状态的变化。因此，人们引入了新的概念——进程，以便从变化的角度，动态地分析研究并发程序的活动。

进程是指具有一定独立功能的程序关于某个数据集合的一次运行活动。为了描述进程，人们将状态信息和与进程相关的信息组织成一个数据结构，这就是进程管理中十分重要的数据结构——进程控制块（PCB）。

进程和程序是既有联系又有区别的两个概念，它们的区别如下：

- 程序是指令的有序集合，是一个静态概念；进程是程序在处理机上的一次执行过程，是一动态概念。程序可以作为一种软件资料长期保存，而进程则是有一定生命期的，它能够动态地产生和消亡。即进程可由“创建”而产生，由调度而执行，因得不到资源而暂停，以致最后由“撤销”而消亡。
- 进程是一个能独立运行的单位，能与其他进程并行地活动。
- 进程时竞争计算机系统有限资源的基本单位，也是进行处理器调度的基本单位。

进程和程序又是有联系的。在支持多任务运行的操作系统中，活动的最小单位是进程。进程一定包含一个程序，因为程序是进程应完成功能的逻辑描述；而一个程序可以对应多个进程。如果同一程序同时运行在若干不同的数据集合上，它将属于不同的进程。

2.进程竞争与合作

竞争系统资源：进程间的相互制约关系，有一种情况是由于竞争资源而引起的间接的相互制约关系。进程共享系统资源，他们对共享资源的使用是通过操作系统的资源管理程序来协调的。凡需使用共享资源的进程，先向系统提出申请，然后由资源管理程序根据资源情况，按一定的策略来实施分配。

进程协作：当进程之间存在有共享数据时，将引起直接制约关系。例如，并发进程之间共享了某些数据、变量、队列等，为了保证数据的完整性，需要正确地处理进程协作的问题。解决进程协作问题的方法是操作系统提供一种同步机构。各进程利用这些同步机构来使用共享数据，实现正确的协作。进程在以下两种情况需要协作。

- 信息共享。由于多个用户可能对同样的信息感兴趣，所以操作系统必须提供支持，以允许对这些资源类型的并发访问。由于对这些信息（或数据）的共享，这些进程是合作进程。
- 并行处理。如果一个任务在逻辑上可分为多个子任务，这些子任务可以并发执行以加快该任务的处理速度。由于这些子任务是为了完成一个整体任务而并发执行的，它们之间一定有间接的相互制约关系，这些进程称之为合作进程。

3.线程

为了进一步提高系统的并行处理能力，在现代操作系统中，使用了一个叫线程（threads）的概念。多线程的概念首先是在多处理机系统的并行处理中提出来的。在操作系统中，为了支持并发活动，引入了进程的概念，在传统的操作系统中，每个进程只存在一条控制线程和程序计数器。但在现代操作系统中，有些提供了对单个进程中多条控制线索的支持。这些控制线索通常称为线程（threads），有时也称为轻量级进程（lightweight processes）。

线程是比进程更小的活动单位，它是进程中的一个执行路径。一个进程可以有多条执行路径即线程。这样，在一个进程内部就可以有多个独立活动单位，可以加快进程处理速度，进一步提高系统的并行处理能力。

线程可以这样描述：

- 线程是进程中的一条执行路径
- 它有自己私有的堆栈和处理机执行环境（尤其是处理器寄存器）
- 它共享父进程的主存
- 它是单个进程所创建的许多个同时存在的线程中的一个

进程和线程既有联系又有区别，可以进一步将进程概括为以下几个方面：

- 一个可执行程序，它定义了初始化代码和数据
- 一个私用地址空间（address space），它是进程可以使用的一组虚拟主存地址
- 进程执行时所需的系统资源（如文件、信号灯、通信端口等）是由操作系统分配给进程的
- 若系统支持线程运行，那么，每个进程至少有一个执行线程

进程是任务调度的单位，也是系统资源的分配单位；而线程是进程中的一条执行路径，当系统支持多线程处理器时，线程是任务调度的单位，但不是系统资源的分配单位。线程完全继承父进程占有的资源，当它活动时，具有自己的运行现场。

4.进程调度

处理器分配是由调度程序来完成的，由于调度本身也要消耗处理机时间，因此不能过于频繁地进行。为此，许多操作系统把调度工作分为两级进行。对于较低一级的调度工作可以较为频繁地进行（如，每隔几十毫秒进行一次），且只需考虑一小段时间内的情况，所以算法简单，调度所花时间也较少，这一级称为低级调度或短程调度，也就是进程调度。对于较高一级的调度，需考虑的因素较多，算法比较复杂，所以进行的次数应较少一些。这种调度称为高级调度或中程调度。在批处理系统中，当某作业撤离或新作业进入时，选择一作业进入主存的调度属此种调度。

虽然对处理机的分配分两级进行的，但制定这两级调度的原则应该是一致的，即都必须符合系统总的设计目标。

（1）进程优先数调度算法

进程优先数调度算法是一种优先调度算法，该算法预先确定各进程的优先数，系统将处理机的使用权赋予就绪队列中具备最高优先级（优先数和一定的优先级相对应）的就绪进程。这种算法又可分为不可抢占CPU与可抢占CPU两种情况。在后一种情况下，无论何时，执行着的进程的优先级总要比就绪队列中的任何一进程的优先级高。

（2）循环轮转调度

常用的进程调度策略除了优先调度策略外，还有一种策略是先来先服务（First In First Out，FIFO）策略，这种策略按提出请求的先后次序排序。一个进程转为就绪状态时加入就绪队列末端，而调度则从队首选取进程。采用这种调度算法可能存在一个问题，一个进程在放弃处理机的控制权之前可能执行很长时间，即它将长时间占用处理机，而使其他进程的推

进收到言重影响。特别是当系统的设计目标是希望用户能得到公平的响应时，这种调度策略是极其不合适的。为此，提出循环轮转调度算法，系统规定一个时间片，每个进程被调度时分得一个时间片，当这一时间片用完时，该进程转为就绪状态并进入就绪队列末端，这种算法遵循循环轮转（ROUND-ROBIN）规则。

当CPU空闲时，选取就绪队列首元素，赋予时间片。当该进程时间片用完，则释放CPU控制权，进入就绪队列的队尾，CPU控制权给下一个处于就绪队列首元素。

如果就绪队列有k个就绪进程，时间片的长度为q秒，则每个进程在每$k*q$的时间内可获得q秒的CPU时间，所以就绪队列的大小决定进程以什么样的速度推进的一个重要因素。此外，时间片q也是一个十分重要的因素，时间片q的计算公式为：$q=\frac{t}{n_{max}}$。

其中：t为用户能接受的响应时间，n为进入系统的进程的最大数目。如果q取得太大，使所有进程都能在分给它的时间片内执行完毕，则此时的轮转算法已经退化为FIFO算法；若q选择适中，则将使得就绪队列中的所有进程都能得到同样的服务；但当q取得很小时，由一个进程到另一个进程的切换时间就变得不可忽略，换言之，过小的q值会导致系统开销的增加。通常q取100ms或更大。

（3）多级反馈队列调度

多级反馈队列调度（multilevel feedback queue scheduling）算法采用多就绪队列结构，多个就绪队列时这样组织的：每个就绪队列的优先级按序递减，而时间片的长度则按序递增。亦即处于序数较小的就绪队列中的就绪进程的优先级要比序数较大的队列中的就绪进程的优先级高，但它获得的CPU时间片要比后者短。

进程从等待状态进入就绪队列时，首先进入序数较小的队列中；当某进程分到处理机时，就给它一个与就绪队列对应的时间片；该时间片用完时，它被迫释放处理机，并进入到下一级（序数增加1，对应时间片也增加1倍）的就绪队列中；虽然它重新执行的时间被推迟了一些，但在下次得到处理机时，时间片却增加了一倍；当处于最大的序数就绪队列时，时间片可以无限大，即一旦分得处理机就一直运行结束。

当CPU空闲时，首先从序号最下的就绪队列查找，取队列首元素去运行，若该就绪队列为空，则从序号递增的下一个就绪队列选进程运行。如此类推，序号最大的就绪队列中的进程只有在其上所有队列都为空时，才有机会被调度。

由此可见，这种算法可以先用较小的时间片处理完那些用时较短的进程，而给那些用时较长的进程分配较大的时间片，以免较长的进程频繁被中断而影响处理机的效率。

多级反馈队列调度时较通用的CPU调度算法，但它也是较复杂的算法。

5.进程间通讯的方式

- 管道（pipe）：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用，进程的亲缘关系指父子进程关系。
- 有名管道（named pipe）：有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。
- 信号量（semaphore）：信号量是一个计数器，它可以控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间同步手段。
- 消息队列（message queue）：消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字符流以及缓冲区大小受限等缺点。
- 信号（signal）：信号是一种比较复杂的通信方式，用于通知接收进程某个时间已经发生。

- 共享内存（shared memory）：共享内存就是映射一段能够被其他进程所访问的内存，这段内存由一个进程创建，但多个进程都可以访问。共享内存时最快的IPC方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量，配合使用，实现进程间的同步与通信。
- 套接字（socket）：套接字也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

二、主存管理——淘汰机制与策略

当程序运行中需要调入新页面而所分得的主存快已用完时，需要淘汰一页。系统应提供淘汰机制和淘汰策略，包括扩充页表数据项、确定页面淘汰原则和是否需要淘汰页面的判断及处理。请求分页系统的指令执行过程中，页面请调和页面淘汰随时可能发生，并能自动地处理。

1.页面淘汰机制

页面淘汰，又称页面置换。为了给置换页面提供依据，页表中还必须包含关于页面的使用情况的信息，并增设专门的硬件和软件来考查和更新这些信息。这说明页表的功能还必须进一步扩充。于是，在页表中增加引用位和改变位。引用位是用来指示某页最近被访问过没有：为“0”表示没有被访问过；为“1”表示已被访问过。改变位是表示某页是否被修改过：为“1”表示已被修改过；为“0”表示未被修改过。这一信息是为了在淘汰一页时决定是否需要写回辅存而设置的。因此，这种情况下完整的页表结构在逻辑上至少包含这几个数据项：页号、主存块号、中断位、改变位、引用位、辅存地址。

2.置换算法

在分页存储管理方法中，主存被等分成一系列的块，程序的地址空间被等分成一系列的页面，然后将页面存放到主存块中。为了便于实现动态地址变换，主存的块和页面大小相等并为2的幂次。

当请求程序要调进一个页面，而此时该程序所分得的主存快已全部用完，则必须淘汰该程序已在主存中的一个页。这时，就产生了在诸页面中淘汰哪个页面的问题，这就是淘汰算法或称为置换算法。

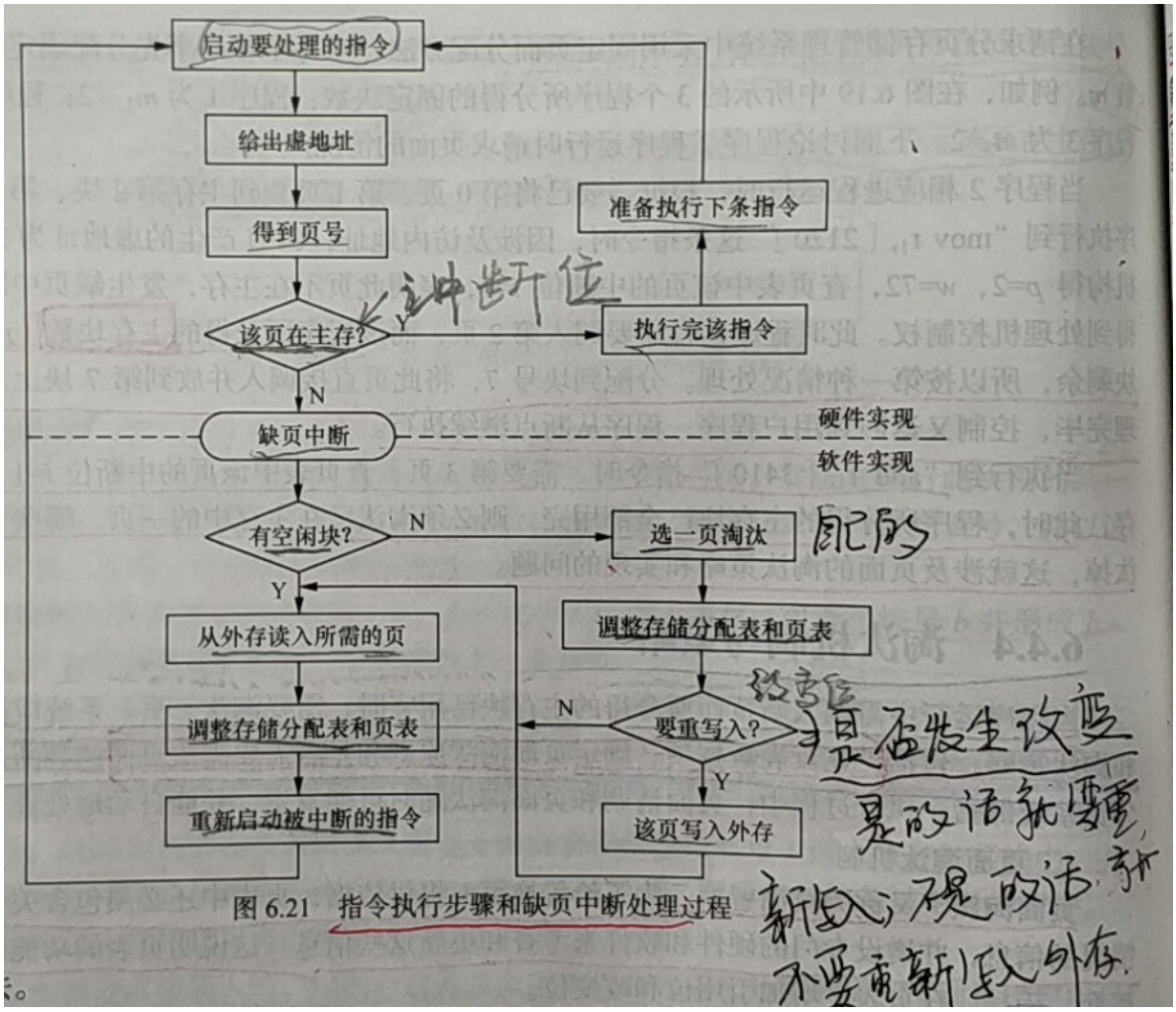

图 6.21　指令执行步骤和缺页中断处理过程

置换算法可以描述如下：当要索取一页面并送到主存时，必须将该程序已在主存中的某一页面淘汰掉。用来选择淘汰哪一页的规则就叫做置换算法。

3.颠簸（抖动）

如果选择的淘汰算法不好，将会使程序执行过程中请求调页的频率大大增加，甚至可能出现这样的现象：刚被淘汰出去的页，过后不久又要访问它，因而又把它调入，而调入后不久又再次被淘汰，再访问再调入，如此反复，使得整个系统的页面置换非常频繁，以致大部分的机器时间花费在来回进行页面的调度上，只有一小部分时间用于程序的实际运行，从而直接影响整个系统的效率。这种现象称为系统抖动。

导致系统效率急剧下降的主存和辅存之间的频繁页面置换现象称为颠簸（thrashin），又可称为抖动。

4.几种页面置换算法

当出现缺页中断时才会面临置换。

（1）最佳算法（OPT算法）

最佳算法是一个理论算法。假定程序p共有n页，而系统分配给它的主存只有m块，即最多只能容纳m页（$1 \leq m \leq n$）。并且，以程序在执行过程中所进行的页面置换次数多寡，即页面置换频率的高低来衡量一个算法的优劣。在任何时刻，若所访问的页已在主存，则再次方位成功；若访问的页不在主存，则称此次访问失败，并产生缺页中断。如果程序p在运行中成功的访问次数为s，不成功的访问次数为f，那么，总的访问次数$a = s + f$。

若定义$f' = \frac{f}{a}$，则称f'为缺页中断率。显然f'和主存固定空间大小m、程序p本身以及调度算法r有关，即$f' = f(r, m, p)$。最佳算法是指对于任何m和p，有$f(r, m, p)$最小。从理论上说，最佳算法是当要调入一新页面而必须先淘汰一旧页时，所淘汰的那一页应是以后不再要用的，或者是在最长的时间以后才会用到的页。然而这样的算法是无法实现的。

（2）先进先出淘汰算法（FIFO算法）

先进先出算法的实质是，总是选择在主存中居留时间最长（即最老）的一页淘汰。其理由是最早调入主存的页，其不再被使用的可能性比最近调入主存的可能性大。这种算法实现起来比较简单，只要系统保留一张次序表即可。

先进先出算法较容易实现，对于具有按线性顺序访问地址空间的程序是比较合适的，而对其他情况则效率不高。因为，那些常常被访问的页，可能在主存中停留得最久，结果这些常用的页终因变“老”而不得不被淘汰出去。据统计，采用这种算法时，缺页中断率差不多是最优算法的三倍。

（3）最久未使用淘汰算法（LRU算法）

最久未使用淘汰算法（least recently used， LRU）的实质是，当需要置换一页时，选择最长时间未被使用的那一页淘汰。LRU算法基于这种理论：如果某一页被访问了，它很可能马上还要被访问；相反，如果它很长时间未曾用过，看起来在最近的未来也不大需要的。实现真正的LRU算法是比较麻烦的，它必须登记每个页面上次访问以来所经历的时间，当需要置换一页时，选择时间最长的一页淘汰。

LRU淘汰算法被认为是一个很好的淘汰算法。主要问题是如何实现LRU置换，为了精确地实现这一算法，要为访问的页面排一个序，该序列按页面上次使用以来的时间长短排序。有两种可行的方案：

- 计数器。用硬件实现最久未淘汰算法，需要为每个页面项关联一个使用时间域，并为CPU增加一个逻辑始终或称为时钟计数器。对每次主存的引用，计数器都会增加。并且时钟计数器的内容要复制到相应页所对应页表项的时间域。当需要置换一页时，选择具有较小时间的页。这种方案需要搜索页表以查找时间域，并且每次主存访问都要写主存（写到页表的时间域）。在任务切换时（因CPU调度）也必须保持时间，还要考虑时钟溢出问题。
- 页号堆栈。LRU淘汰算法也可以使用软件来实现，建立页号堆栈来登记进程已进入主存的页号，且按最近被访问的次序来排序。栈顶是最近访问的页号，栈低是最久未使用的页号。当一个页面被访问过，就立即将它的页号记在页号栈的顶部，而将栈中原有的页号依次下移。如果栈中原有页号有与新记入顶部的页号相重者，则将该重号抽出，且将页号栈内容进行紧凑压缩。这样，栈低存放的页号，就是自上次访问以来最久未被访问过的页号，该页应先被淘汰。
- LRU近似算法。该算法只要求每一个存储块有一个“引用位”，当某块中的页面被访问时，这一位由硬件自动置“1”，而页面管理软件周期性（设周期为T）地将所有引用位重新置“0”。这样，在时间T内，某些被访问过的页面，其对应的引用位为1，而未被访问过的页面，其对应的引用位为0。因此，可以根据引用位的状态来判断各个页面最近使用的情况。当需要置换一页时，选择引用位为0的页并淘汰。

页号堆栈方法示例：

有一个页面访问序列“3 5 0 5 1 0 8 2 5 1 3”，该程序分得的主存块数为5，则构建5个单元的栈。

3 5 0： 入栈之后的的顺序为（从栈低到栈顶）3 5 0

随后5入栈之后的结果为（从栈低到栈顶）： 3 0 5

随后1入栈之后的结果为（从栈低到栈顶）：３０５１

随后0入栈之后的结果为（从栈低到栈顶）：３５１０

随后8入栈之后的结果为（从栈低到栈顶）：３５１０８

随后2入栈之后的结果为（从栈低到栈顶）：５１０８２，此时3已被淘汰

随后5入栈之后的结果为（从栈低到栈顶）：１０８２５

随后1入栈之后的结果为（从栈低到栈顶）：０８２５１

随后3入栈之后的结果为（从栈低到栈顶）：８２５１０，此时0已被淘汰

缺页中断次数计算：

有一个页面访问序列“０３１０４０５”，该程序分得的主存块数为3。