第5章 处理器调度

处理器是计算机系统中一个十分重要的资源。随着多道程序设计技术的出现，处理器的管理也日趋复杂。对于不同类型的操作系统，处理器的管理方法各不相同。不同的处理器管理方法将为用户提供不同性能的操作系统。因此，根据操作系统目标的不同，处理器的管理策略也不尽相同。

本章将以处理器管理为核心，讨论与处理器调度有关的概念，并介绍常用的调度算法。

5.1 三级调度的概念

在多道程序系统中，一个作业被提交后，并不一定能立即执行，必须经过处理器调度，才可为其创建进程、分配内存，从而进入执行状态，这个过程被称为作业调度或高级调度。而作业的执行归结为进程的调度，又称为低级调度。在比较完善的系统中，为了提高内存的利用率，往往还设置了对换调度，即中级调度。本节将详细介绍处理器的三级调度。

5.1.1 作业的状态及其转换

作业是用户在一次解题或一个事务处理过程中要求计算机系统所做的工作的集合。通常情况下，作业由用户程序、所需的数据及作业说明书3部分组成。程序是问题求解的算法描述；数据是程序加工的对象，但有些程序未必使用数据；作业说明书是要求操作系统对作业的程序和数据进行何种控制进行说明的程序。

计算机系统在完成一个作业的过程中所做的一项相对独立的工作称为一个作业步。因此，也可以说，一个作业是由一系列有序的作业步组成的。例如，在编制程序的过程中，通常要进行编辑输入、编译、链接、运行等几个步骤，其中的每一个步骤都可以作为一个作业步。

在多道程序系统中，一个作业在它的生命期内要经历提交、后备、运行和完成4个主要阶段，一般要由系统经过多级调度才能实现，故相应地，作业就有4种状态。

1. 提交状态

当用户将自己的作业提交给操作员，操作员通过某种输入方式将用户提交的作业输入到外存上时，称作业处于提交状态。作业输入方式通常有5种。

(1) 联机输入方式：该方式大多用在交互式系统中，用户和系统通过交互式会话来输入作业。在联机输入方式中，外围设备直接和主机相连接，一台主机可连接一台或多台外围设备。

(2) 脱机输入方式：该方式又称预输入方式。主要是为了解决单台设备联机输入时的处理器浪费问题。脱机输入方式利用低档I/O处理器作为外围处理器进行输入处理，提高

了主机的资源利用率。脱机输入方式的缺点是灵活性差,当遇到紧急任务需要处理时,无法直接交给主机优先处理。

(3) 直接耦合方式:该方式是把主机和外围低档机通过一个公用的大容量外存直接耦合起来,从而省去了在脱机输入时依靠人工干预来传递给后援存储器的过程。在该方式中,慢速的输入过程仍由外围机管理,而对公用存储器的高速读写则由主机完成。

(4) SPOOLing 系统:在 SPOOLing 系统中,多台外围设备通过通道或 DMA 器件将主机与外存连接起来,作业的输入过程由主机中的操作系统控制。在系统输入模块收到作业输入请求信号后,输入管理模块中的读进程负责将信息从输入装置读入缓冲区。当缓冲区满时,由写过程将信息从缓冲区写到外存输入井中。读、写过程反复循环,直到一个作业输入完毕。当读进程读到一个结束标志之后,系统再次驱动写进程把最后一批信息写入外存并调用中断处理程序结束该次输入。SPOOLing 系统的有关概念详见第 9 章。

(5) 网络输入方式:该方式以上述几种输入方式为基础。当用户需要把在网络中某一台主机上输入的信息传递到同一网络中的另一台主机上进行操作或执行时,就构成了网络输入方式。

2. 后备状态

后备状态也称收容状态。输入管理系统不断地将作业输入到外存中相应的部分(或称输入井,即专门用来存放待处理作业信息的一组外存分区)。当操作系统为输入并存放在外存输入井上的作业建立一个作业控制块(Job Control Block,JCB)之后,作业就进入了后备状态。也就是说,作业从建立 JCB 到被作业调度程序选中运行所处的状态称为后备状态。

3. 运行状态

作业调度程序从处于后备状态的作业队列中,按一定的算法选中一个作业调入内存,并为之建立相应的进程后,由于此时的作业已具有独立运行的资格,如果处理器空闲,便可立即开始执行,称此时的作业进入了运行状态。处于运行状态的作业在系统中可以从事各种活动。它可能被进程调度程序选中而在处理器上执行;也可能在等待某种事件或信息;还有可能在等待进程调度程序为其分配处理器。因此,从宏观上看,作业一旦由作业调度程序选中进入内存就开始了运行,但从微观上讲,内存中的作业并不一定正在处理器上运行。为了便于对运行状态的作业进行管理,根据进程的活动又把进程分成就绪、执行和阻塞 3 个基本状态。

4. 完成状态

当作业(进程)运行正常完成或异常结束时,便自我终止(正常完成),或被迫终止(异常结束),此时作业便进入完成状态。处于完成状态的作业被作业终止程序回收其作业控制块及已分配给它的所有资源,然后作业随之消亡。

图 5.1 表示作业在整个生命活动期间的状态及其转换。作业由提交状态到后备状态的转换,是由作业建立程序完成的;从后备状态转变为运行状态是由调度程序所引起的;而作业由运行状态自愿或被迫地转变为完成状态,则是在有关作业终止的系统调用的作用下完成的。

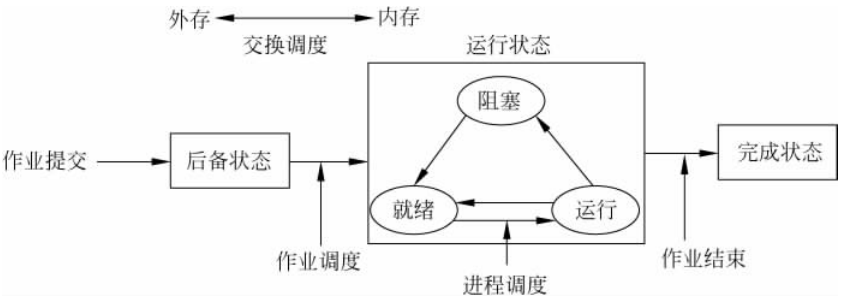


图 5.1　作业的状态及其转换

5.1.2　调度的层次

处理器的调度问题实际上也是处理器的分配问题。显然，只有那些参与竞争处理器所必需的资源都已得到满足的进程才享有竞争处理器的资格。这时，它们处于就绪状态。这些必要的资源包括内存、外设及有关数据结构(如 PCB)等。在进程有资格竞争处理器之前，作业调度程序必须先调用存储管理、外设管理程序，并按一定的选择顺序和策略选择出若干个处于后备状态的作业，为它们分配内存等资源并创建进程，使它们获得竞争处理器的资格。

另外，由于处于运行状态的作业一般包含有多个进程，而在单处理器系统中，每一时刻只能有一个进程占用处理器。那么，其他进程就只能处于准备抢占处理器的就绪状态或等待得到某种新资源的阻塞状态。为了提高资源的利用率，有些操作系统把一部分在内存中处于就绪状态或阻塞状态而在短时期内又得不到执行的进程换出内存，以让其他作业的进程竞争处理器。故一般地，按调度的层次，可将调度分为 3 级。

1. 高级调度

高级调度又称作业调度。作业调度程序决定把哪些后备队列中的作业调入内存，并为其创建进程，分配必要的资源，最后将所创建的进程插入就绪队列，以使该作业的进程获得竞争处理器的权利。在批处理系统中或者是操作系统中的批处理部分都配有作业调度。作业调度的运行频率较低，如几分钟才调度一次，且调度算法复杂。

2. 中级调度

中级调度又称交换调度，主要功能是按一定的算法在内存和外存之间进行进程对换，其目的是缓解内存紧张的情况。交换调度的主要工作是将内存中处于阻塞状态的某些进程换至外存，腾出内存空间以便将外存上已具备运行条件的进程换入内存，准备运行。进程在其整个生命期间可能要经历多次换入换出，在分时系统中常采用交换调度。

3. 低级调度

低级调度又称进程调度。其主要任务是按照某种策略和方法选取一个处于就绪状态的进程占用处理器。它决定就绪队列中哪个进程先获得处理器，然后再由分派程序执行将处理器分配给进程的操作。进程调度的运行频率很高，典型情况是几十毫秒一次。进程调度是最基本的调度，在 3 种类型的操作系统中都必须配置它。图 5.2 所示是一种简单的调度模型。

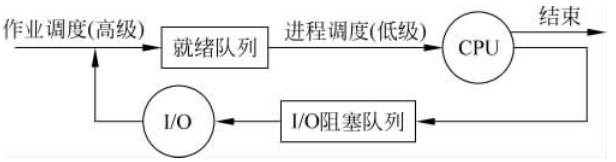


图 5.2　简单的排队调度模型

5.1.3　调度模型

从上述操作系统的调度类型可知,虽然操作系统的调度机制不尽相同,有的操作系统仅设有低级调度,有的操作系统则拥有高级调度和低级调度,有的操作系统三级调度一应俱全。但是操作系统中的任何一种调度都涉及进程队列,因此模拟操作系统的不同性能,一般有 3 种相应的调度队列模型。

1. 一级调度队列模型

一级调度模型仅设置了进程调度。在这种调度模型中,用户输入的命令和数据都直接送入内存。操作系统会为每一个作业建立一个或多个进程,并将它们插入就绪队列,然后按照时间片轮转方式进行调度。一级调度队列模型如图 5.3 所示。

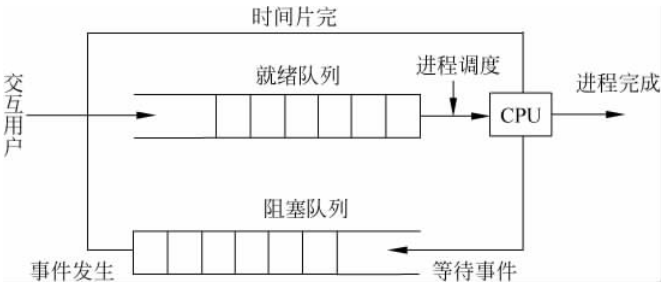


图 5.3　一级调度队列模型

2. 二级调度队列模型

在二级调度队列中不仅有进程调度,而且还有作业调度。作业调度从外存中选择一批作业调入内存,为其创建进程并送入就绪队列。同时,当操作系统任务较多时,阻塞队列有可能很长。为了提高执行效率,通常设置若干个阻塞队列,不同队列对应于引起进程阻塞的不同原因。二级调度队列模型如图 5.4 所示。

3. 三级调度队列模型

三级调度队列模型同时拥有作业调度、交换调度和进程调度。在引入交换调度后,可以把处于静止就绪和静止阻塞状态的进程从内存交换到外存上,以使内存紧张的状况得以缓解。三级调度队列模型如图 5.5 所示。

5.1.4　作业和进程的关系

一个作业可被看作是用户向计算机提交的一个任务实体,如一次计算、一个控制过程等。而进程则是计算机为完成用户所提交的任务实体而设置的执行实体,是系统分配资源的基本单位。显然,计算机要完成一个任务实体,必须要有一个以上的执行实体。一个作业

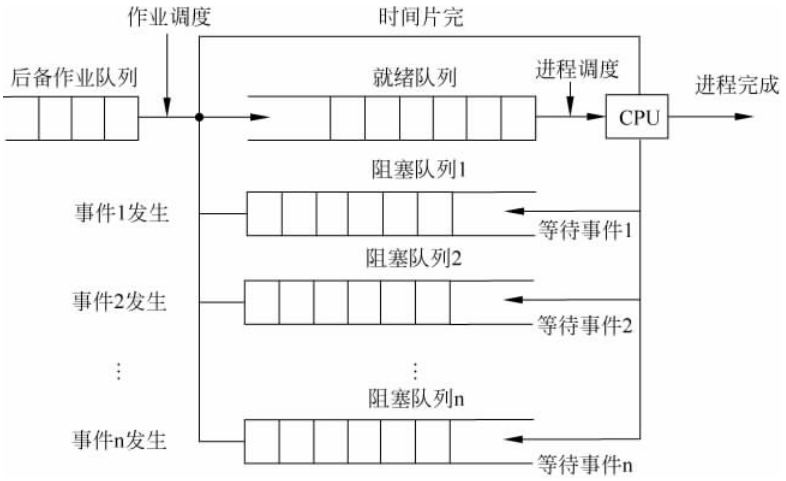


图 5.4　二级调度队列模型

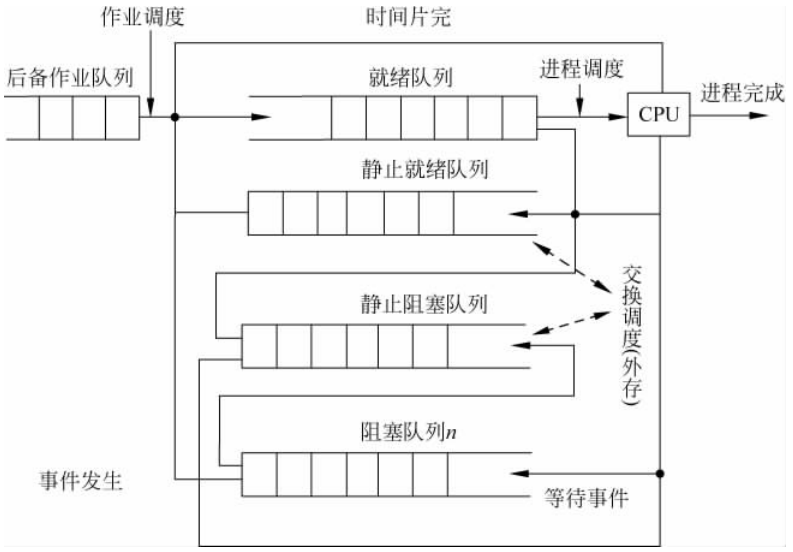


图 5.5　三级调度队列模型

被调度运行之后，系统便要为它创建进程。一般来说，为一个作业创建一个进程，该进程也称为根进程，然后，在执行作业控制语句时，根据任务要求，系统或根进程为作业创建相应的子进程，并为各子进程分配资源和调度各子进程运行以完成作业所要求的任务。

5.2　作 业 调 度

只有批处理系统才必须具有作业调度，在分时系统和实时系统中都没有作业调度的概念。由于在分时系统中要进行人机交互，系统必须能及时响应，为此应把用户从终端输入的作业直接送入内存而不是建立在外存上，因此，不再需要专门用于把作业从外存调入内存的作业调度过程。对于实时系统中的实时任务，因为通常对其响应的时间要求更为严格，故不

需要作业调度。

5.2.1 作业调度的功能

作业调度主要是完成作业从后备状态到运行状态的转变,以及从运行状态到完成状态的转变。其主要功能如下。

1. 记录系统中各作业的状况

作业调度程序要能挑选出一个作业投入运行,并且在运行中对其进行管理。它必须掌握作业在各个状态,包括运行阶段的有关情况。通常,系统为每个作业建立一个作业控制块JCB来记录这些相关信息。与系统感知进程存在时要通过进程控制块一样,系统通过JCB来感知作业的存在。作业在各阶段所要求和分配的资源以及作业的状态都记录在它的JCB中后,作业调度程序根据JCB中的有关信息,对作业进行调度和管理。

对于不同的批处理系统,其JCB的内容也有所不同。JCB的主要内容包括作业名、作业类型、资源要求、当前状态、资源使用情况以及该作业的优先级等。

其中,作业名由用户提供并由系统将其转换为系统可识别的作业标识符。作业类型指该作业属于计算型(要求CPU时间多)还是I/O型(要求输入/输出量大),或图形设计型(要求高速图形显示)等。而资源要求则包括该作业估计执行时间、要求最迟完成时间、要求的内存量和外存量、要求的外设类型及台数以及要求的软件支持工具库函数等,资源要求均由用户提供。资源使用情况包括作业进入系统的时间、开始运行时间、已运行时间、内存地址、外设台数等。作业进入系统的时间是指作业的全部信息进入输入井,作业的状态成为后备状态的时间。开始运行时间指该作业被调度程序选中,其状态由后备状态变为运行状态的时间。内存地址指分配给该作业的内存区起始地址。外设台数指分配给该作业的外设实际台数,优先级则被用来决定该作业的调度次序。优先级既可以由用户给定,也可以由系统动态计算产生。状态是指作业当前所处的状态。显然,只有当作业处于后备状态时才可以被调度。

2. 从后备队列中挑选出一部分作业投入执行

一般来说,系统中处于后备状态的作业较多,大的系统可以达到几十个甚至几百个,这取决于输入井的空间大小。但是,处于运行状态的作业一般只有有限的几个。作业调度程序根据选定的调度算法,从后备作业队列中挑选出若干作业投入运行。

3. 为被选中的作业做好运行前的准备工作

作业调度程序为选中的作业建立相应的进程,并为这些进程分配所需要的全部资源,如为其分配内存、外存、外设等。

4. 在作业运行结束时做善后处理工作

善后处理工作主要是输出作业管理信息,例如执行时间等。另外还包括回收该作业所占用的资源,撤销与该作业有关的全部进程和该作业的作业控制块等。

5.2.2 作业调度的目标与性能衡量

操作系统对作业进行调度的目标是使系统获得尽可能高的效率,这就需要选择合适的调度方式和算法。但从系统的角度和从用户的角度出发,衡量操作系统好坏的准则是不一样的。因此,此准则分为面向系统的准则和面向用户的准则两类。

1. 面向系统的准则

对不同的操作系统有不同的要求，为了满足系统功能的要求，在设计操作系统时，应遵循一些准则，最重要的有以下几点。

1）吞吐量大

这是用来评价批处理系统的重要指标。吞吐量是指单位时间内完成的作业数，与批处理作业的平均长度有着密切关系。但作业调度的方式和算法，也将对吞吐量的大小产生较大的影响。对于同一批作业，若选用好的调度方式和算法，可获得大得多的系统吞吐量。

2）CPU 利用率高

对于大中型多用户系统，由于 CPU 价格十分昂贵，所以 CPU 的利用率成为衡量大中型系统性能的一个十分重要的指标，而调度方式和算法，对 CPU 的利用率起着十分重要的作用。在实际计算机系统中，CPU 的利用率在 40%～90%之间。该准则对于单用户微机或某些实时系统而言，就不那么重要。

3）各类资源的平衡利用

在大中型系统中，有效地利用各类资源（如 CPU、内存、外存和 I/O 设备等），也是一个重要指标。对于微机和某些实时系统，该准则也不那么重要。

2. 面向用户的准则

为了满足用户对操作系统的要求，应遵循如下准则：

1）周转时间短

周转时间是评价批处理系统的一个重要性能指标。作业周转时间是指从作业提交给系统开始，到作业完成为止的时间间隔。可以发现，作业提交后不一定能马上运行，从作业提交到作业开始运行的这段时间是等待时间。作业的等待时间越短，则周转时间越短。

2）平均周转时间

计算机系统为使大多数用户对周转时间感到满意，引入了平均周转时间 T 的评价指标。对于有 n 个作业的系统，平均周转时间可描述为：

$$T = \frac{1}{n}\sum_{i=1}^{n} T_i = \frac{1}{n}\sum_{i=1}^{n}(Te_i - Ts_i)$$

式中，n 是系统中的作业个数，T_i 是第 i 个作业的周转时间，Te_i 是作业的完成时间，Ts_i 是作业的提交时间。

3）平均带权周转时间

周转时间没有考虑作业的运行时间，不能准确反映作业周转时间与作业运行时间的关系，为此引入了平均带权周转时间。一个作业的带权周转时间定义为作业的周转时间与系统为它提供的实际服务时间之比，即 $W = T_i / Tr_i$。

n 个作业的平均带权周转时间为：

$$W = \frac{1}{n}\sum_{i=1}^{n} \frac{T_i}{Tr_i}$$

式中，Tr_i 是系统为作业 i 提供的实际服务时间。

4）响应时间短

响应时间是评价分时系统的性能指标。它是指从用户通过键盘提交一个请求开始，直至系统首次产生响应为止的时间，或者说直到屏幕显示结果为止的时间。

5) 截止时间的保证

在实时系统中,截止时间是衡量系统性能好坏的重要指标。截止时间是指某任务必须完成的最迟时间。对于严格的实时系统,其调度方式和调度算法必须保证这一点,否则可能引起难以预料的后果。

6) 优先权准则

在批处理、分时、实时和多模式系统中,都可使用优先权准则,以便让那些紧急的作业得到及时处理。在要求较严格的场合,往往还需要选用抢占调度方式,才能保证紧急作业得到及时处理。

5.3 进 程 调 度

进程调度的任务是控制、协调多个进程对处理器的竞争,即按照一定的调度算法从就绪队列中选中一个进程,并把处理器的使用权交给被选中的进程。操作系统为了对进程进行有效的监控,需要维护一些与进程相关的数据结构,记录所有进程的运行情况,并在进程让出处理器或调度程序剥夺运行状态进程占用处理器时,选择适当的进程分派处理器,完成上下文的切换。我们把操作系统内核中完成这些功能的部分称为进程调度程序(process scheduler),它所使用的算法称为调度算法(scheduling algorithm)。

5.3.1 进程调度的功能

进程调度是操作系统的最基本功能之一,其具体功能可总结为如下几点。

1. 记录系统中所有进程的执行情况

作为进程调度的准备工作,进程管理模块必须将系统中各进程的执行情况和状态特征记录在各进程的PCB中,作为管理进程的依据。同时根据各进程的状态特征和资源需求等,将各进程的PCB表排成相应的队列并进行动态队列转接。进程调度模块通过PCB的变化来掌握系统中所有进程的执行情况和状态特征,并在适当的时机从就绪队列中选择一个进程占有处理器。

2. 选择占有处理器的进程

进程调度的主要功能是按照一定的策略选择一个处于就绪状态的进程,使其获得处理器运行。根据不同的系统设计目标,有各种各样的选择策略,例如系统开销较少的静态优先数调度法、适合于分时系统的时间片轮转法和多级反馈队列法等。这些选择策略决定了算法的性能。

3. 进行进程上下文的切换

进程的上下文是进程执行活动全过程的静态描述。一个进程的上下文包括进程的状态、有关变量和数据结构的值、机器寄存器的值以及有关程序等。一个进程的执行是在进程的上下文中进行的。当正在运行的进程由于某种原因要让出处理器时,系统要做进程上下文切换,以使另一个进程得以运行。当进行上下文切换时,系统首先要检查是否允许做上下文切换(在有些情况下,上下文切换是不允许的,例如系统正在执行某个不允许中断的原语时)。然后,要保留有关被切换进程的足够信息,以便以后切换回该进程时,可以顺利恢复该进程的执行。在系统保留了CPU现场之后,调度程序选择一个新的处于就绪状态的进程,

并切换该进程的上下文，使 CPU 的控制权掌握在被选中的进程手中。

5.3.2 进程调度方式

根据进程所要完成任务的不同，进程的类型也各不相同。输入/输出型进程往往要花费很多时间等待 I/O，而计算型进程则在条件允许的情况下连续使用 CPU 很长时间。因此，进程调度程序所考虑的一个问题是：当调度程序启动运行某些进程时，根本不知道进程在阻塞前要运行多久，阻塞可能是因为 I/O、信号量或其他定期发出的中断，通常每秒 50～60 次。但在许多计算机上，操作系统能根据需要将时钟频率设置成任意值。每发生一次时钟中断，操作系统都需决定当前进程是继续运行，还是它已经占用了足够长的 CPU 时间，应该暂停而让其他进程运行。进程调度的方式通常分为非剥夺调度方式和剥夺调度方式两种。

1. 非剥夺调度方式

调度程序一旦把处理器分配给某进程，该进程就将一直运行下去，只有当进程完成或发生其他事件而阻塞时，系统才把处理器分配给另一进程，这样的调度方式称为非剥夺调度方式，也称非抢占方式(nonpreemptive scheduling)。在此调度方式下，系统不得以任何理由剥夺现行进程所占用的处理器。该调度方式的优点是简单、系统开销小，但却可能导致系统性能的恶化，具体表现为：

(1) 当一个紧急任务到达时，不能立即投入运行，从而延误时机。

(2) 若干个后到的短进程，需要等待先到的长进程运行完毕后才能运行，致使短进程的周转时间较长。例如，有 3 个进程 P_1、P_2、P_3 先后(但又几乎同时)到达，它们分别需要 20、4 和 2 个单位时间运行完毕。若按它们到达的先后顺序执行，且不可剥夺，则 3 个进程的周转时间分别为 20、24 和 26 个单位时间，平均周转时间是 70/3 个单位时间。这种非剥夺调度方式对短进程 P_2、P_3 而言是不公平的，因为批处理系统的作业一般都不是十分紧急，且不与用户交互，故常采用非剥夺调度方式。

2. 剥夺调度方式

进程的另一种调度方式是剥夺调度方式，或称抢占方式(preemptive scheduling)，是指进程正在运行时，系统可根据某种原则，剥夺已分配给它的处理器，并将处理器再分配给其他进程的一种调度方式。剥夺的原则有：

(1) 优先权原则。优先权高的进程可以剥夺优先权低的进程的运行。

(2) 短进程优先原则。短进程到达后可以剥夺长进程的运行。

(3) 时间片原则。一个时间片运行完后重新调度。

尽管非剥夺调度算法简单且易于实现，但它通常不适用于具有多个竞争的通用系统。但对于专用系统，如一个数据库服务器，主进程在收到请求时启动一个子进程并让其运行直到结束或阻塞则是很合理的。所以，系统目标不同，所采用的调度方式也可能不同。

5.3.3 进程调度的时机

进程调度发生的时机与引起进程调度的原因以及进程调度的方式有关。引起进程调度的原因有以下几类：

(1) 正在执行的进程执行完毕或由于某种错误而异常终止。

(2) 执行中的进程自己调用阻塞原语将自己阻塞起来而进入等待状态(如等待某一事件而事件未发生或调用了 wait 原语而资源不足等)。

(3) 分时系统中的时间片用完。

(4) 在执行完系统调用等系统程序后返回用户进程时,可看作系统进程运行完毕,从而选择一个新的用户进程执行。

(5) 在基于优先级调度的策略时,就绪队列中的某进程的优先级变得高于当前运行进程的优先级,也将引发进程调度。

以上(1)～(4)是在剥夺或非剥夺调度方式下引起进程调度的原因,而第(5)条是在剥夺调度方式下引起调度的原因。

操作系统将会在以上几种原因之一发生的情况下进行进程调度。由于进程调度的使用频率较高,其性能优劣直接影响进程和进程调度的性能,进而影响操作系统的性能。反映作业调度性能优劣的周转时间和平均周转时间在某种程度上反映了进程调度的性能。进程调度性能的衡量方法可分为定性和定量两种:

(1) 在定性衡量方面,首先是调度的可靠性,包括一次进程调度是否可能引起数据结构的破坏等。这要求对调度时机的选择和保存 CPU 现场十分谨慎。另外,简洁性也是衡量进程调度性能的一个重要指标,由于调度程序的执行涉及多个进程并必须进行上下文切换,如果调度程序过于烦琐和复杂,则会付出较大的系统开销。这在用户进程进行系统调用较多的情况下,将会造成响应时间大幅度增加。

(2) 进程调度的定量评价包括 CPU 的利用率评价、进程在就绪队列中的等待时间与运行时间之比等。实际上,由于进程进入就绪队列的随机模型很难确定,而且进程上下文切换等也将影响进程的执行效率,因而对进程调度性能进行分析是很困难的。一般情况下,大多利用模拟或测试系统响应时间的方法来评价进程调度的性能。

5.4 常用的调度算法

调度算法是指根据系统的资源分配策略所规定的资源分配算法。对于不同的系统和系统目标,通常采用不同的调度算法。目前存在多种调度算法,有的算法适用于作业调度,有的算法适用于进程调度,但也有些调度算法,既可用于作业调度,也可用于进程调度。

5.4.1 先来先服务调度算法

先来先服务(First Come First Serve,FCFS)调度算法是一种最简单的调度算法。其基本思想是将用户作业或就绪进程按提交或变为就绪状态的先后次序排成队列,调度时总是选择队首作业或进程投入运行。该算法既可用于作业调度,也可用于进程调度。

在作业调度中采用该算法时,从后备作业队列中选中一个或几个作业,把它们调入内存,为其分配资源,创建进程,并把进程插入就绪队列;在进程调度中采用该算法时,进程进入就绪队列时,总是排在队尾,每次调度时,选中队首进程,为其分配处理器,使之投入执行。

例如,有 3 个进程 A、B、C 先后(但几乎又是同时)进入就绪队列。它们的 CPU 执行时间分别是 21、6 和 3 个单位时间。在非剥夺调度方式下,按 FCFS 算法调度,它们的执行情况如图 5.6(a)所示。

在图 5.6(a)中，进程 A 的周转时间为 21 个单位时间，进程 B 的周转时间为 27 个单位时间，进程 C 的周转时间为 30 个单位时间，它们的平均周转时间为 26 个单位时间。若它们按 C、B、A 的次序到达，如图 5.6(b)所示，则周转时间分别为 30、9、3 个单位时间，平均周转时间为 14 个单位时间。

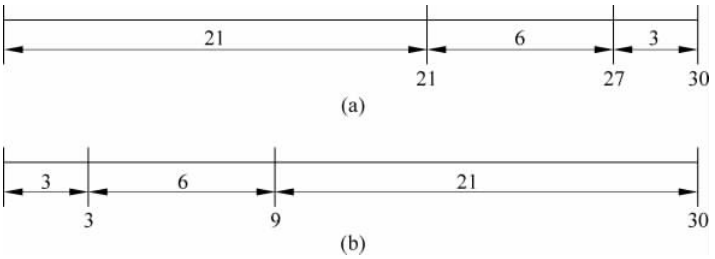


图 5.6　FCFS 调度示例

比较上述两种情况可以看出：虽然 FCFS 调度算法易于实现，表面上也公平，但服务质量不佳，对短作业用户不公平，因此 FCFS 算法很少作为进程调度的主调度算法，而常作为一种辅助调度算法。

扫描二维码，观看视频

【例 5-1】 假设有 5 道作业，它们的进入时间和运行时间如表 5.1 所示。

在非剥夺调度方式的单道环境下，采用 FCFS 调度算法，试说明它们的调度顺序，并计算在该种调度算法下的平均周转时间和平均带权周转时间。

表 5.1　5 道作业的进入时间和运行时间

作　业　号	进入时间/h	运行时间/h
1	0	4
2	1	3
3	2	6
4	3	2
5	4	4

解：采用 FCFS 调度算法，调度顺序是 1、2、3、4、5。

5 道作业的执行时间、完成时间和周转时间如表 5.2 所示。

表 5.2　5 道作业的执行时间、完成时间和周转时间

作　业　号	进入时间/h	运行时间/h	开始执行时间/h	完成时间/h	周转时间/h
1	0	4	0	4	4
2	1	3	4	7	6
3	2	6	7	13	11
4	3	2	13	15	12
5	4	4	15	19	15

平均周转时间 T＝(4＋6＋11＋12＋15)/5＝9.6(h)

平均带权周转时间 W＝(4/4＋6/3＋11/6＋12/2＋15/4)/5＝2.92(h)

5.4.2 短作业(进程)优先调度算法

短作业优先(Shortest Job First,SJF)或短进程优先(Shortest Process First,SPF)调度算法的设计目标是改进FCFS调度算法,减少作业(进程)平均周转时间。该算法要求作业(进程)在开始运行时预计运行时间,对预计运行时间短的作业(进程)优先分派处理器。

SJF调度算法是从后备队列中选择一个或若干个估计运行时间最短的作业,将它们调入内存运行。该调度算法可以照顾到实际上在所有作业中占很大比例的短作业,使它们能比长作业优先执行。

SPF调度算法是从就绪队列中选出一个估计运行时间最短的进程,将处理器分配给它,使它立即运行并一直运行到完成或发生某事件而被阻塞放弃处理器时为止,然后重新调度。

虽然这种算法能有效地降低作业(进程)的平均周转时间、提高系统的吞吐量,但是,它们存在着不容忽视的缺点:

(1) 该算法对长作业(进程)不利。由于系统状态是动态的,如果不断地有短作业(进程)进入,则可能导致长作业(进程)长时间得不到响应。这种情况称为"饥饿现象"。

(2) 该算法未考虑作业(进程)的紧迫程度。

(3) 由于作业(进程)的长短只是根据用户所提供的估计运行时间而定,而用户又可能会有意或无意地缩短其作业(进程)的估计执行时间,致使该算法不一定能真正做到短作业(进程)优先调度。

扫描二维码,观看视频

【例5-2】 仍采用例5-1的题目,在非剥夺调度方式的单道环境下,采用SJF调度算法,试说明它们的调度顺序,并计算在该种调度算法下的平均周转时间和平均带权周转时间。

解:采用SJF调度算法,调度顺序是1、4、2、5、3。

5道作业的执行时间、完成时间和周转时间如表5.3所示。

表5.3 5道作业的执行时间、完成时间和周转时间

作业号	进入时间/h	运行时间/h	开始执行时间/h	完成时间/h	周转时间/h
1	0	4	0	4	4
2	1	3	6	9	8
3	2	6	13	19	17
4	3	2	4	6	3
5	4	4	9	13	9

平均周转时间 T=(4+3+8+9+17)/5=8.2(h)

平均带权周转时间 W=(4/4+3/2+8/3+9/4+17/6)/5=2.05(h)

5.4.3 时间片轮转调度算法

在分时系统中采用时间片轮转法(Round Robin,RR),其基本思想是让每个进程在就绪队列中等待的时间与享受服务的时间成正比。轮转法的基本概念是将CPU的处理时间分成固定大小的时间片,且采用剥夺调度方式。在简单轮转法中,系统将所有就绪进程按先

进先出规则排成一个环形队列，把 CPU 分配给队首进程，并规定它执行一个时间片。当时间片用完时，系统剥夺该进程的运行并将它送到就绪队列末尾，重新把处理器分配给就绪队列中新的队首进程，同样也让它运行一个时间片。这样，就绪队列中的所有进程在一给定时间内，均可获得一个时间片的处理器运行时间。为了实现轮转调度，所有就绪进程的 PCB 排成环形队列并使用一个当前指针扫描它们。

在时间片轮转算法中，时间片 q 的大小对计算机的性能有很大的影响。如果时间片太大，大到每个进程都能在该时间片内执行完毕，则时间片轮转算法便退化为 FCFS 算法，因而无法获得令用户满意的响应时间。如果时间片过小，用户输入的常用命令都要花费几个时间片才能处理完毕，同样难以满足用户对响应时间的要求。因此，时间片的大小应选择适当，通常要考虑的因素如下：

1. 系统对响应时间的要求

作为分时系统，首先必须满足系统对响应时间的要求。由于响应时间 T 直接与用户（进程）数目 n 和时间片 q 成比例，即 T＝nq，因此在用户（进程）数一定时，时间片的长短将正比于系统所要求的响应时间。例如，T＝3s，n＝100 时，q＝30ms。

2. 就绪队列中进程的数目

在分时系统中，就绪队列中所有的进程数，是随着在终端上机的用户数目而改变的，但系统应保证当所有终端都有用户上机时，仍能获得较好的响应时间。因此，时间片的大小应反比于分时系统所配置的终端数目。

3. 系统的处理能力

系统的处理能力是指系统必须保证用户输入的常用命令能在一个时间片内处理完毕，否则将无法得到满意的响应时间，而且会增加平均周转时间及带权周转时间。

【例 5-3】 假设有 5 道作业，它们的进入时间和运行时间如表 5.4 所示。

表 5.4　5 道作业的进入时间和运行时间

作　业　名	进入时间/s	运行时间/s
A	0	4
B	1	2
C	2	5
D	3	3
E	4	3

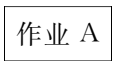
给出时间片分别为 q＝1 和 q＝4 时的调度图，并计算 RR 调度算法下的平均周转时间和平均带权周转时间。

扫描二维码，观看视频

解：当时间片 q＝1 时，可以作出这样的思考：

当时间片处于 0～1s 时，只有作业 A 可以运行。

0～1s：



CPU

当时间片处于 1～2s 时，由于作业 B 已到达，而且基于新进程优先的原则，此时作业 A 让出处理器，进入等待队列，作业 B 运行。

1～2s：

作业 B		作业 A
CPU		等待队列

当时间一到2s，作业B让出处理器，准备进入等待队列，此时的等待队列作业A居于队首，基于新进程优先的原则，新到达的作业C排在作业A之后，而刚刚结束时间片的作业B则排在作业C之后。这个过程的示意图如图5.7所示。

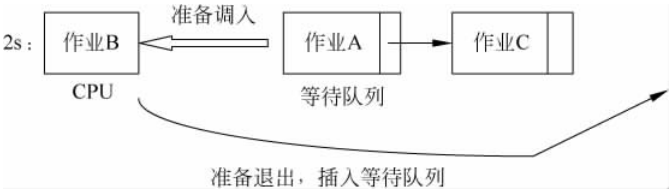


图5.7　作业处理过程

因此时间片处于2～3s时，应该是作业A占用处理器，作业C和B处于等待队列中。

依次分析，用双下画线表示该作业占用CPU，后面的作业处于等待状态，可以写出时间片3～4(s)：C BDA

时间片4～5(s)：B DAEC，剩余的时间片可以由读者自行分析写出。注意当某个作业运行的时间片合计已完成，该作业即结束。

综合以上分析，因此，当时间片q=1时，调度图如图5.8所示。

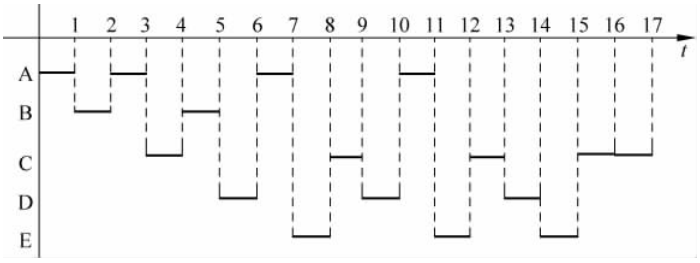


图5.8　q=1时的RR调度算法运行情况

从图5.8得出的5道作业的完成时间和周转时间如表5.5所示。

表5.5　5道作业的完成时间和周转时间

作　业　号	进入时间/s	运行时间/s	完成时间/s	周转时间/s
A	0	4	11	11
B	1	2	5	4
C	2	5	17	15
D	3	3	14	11
E	4	3	15	11

平均周转时间 T=(11+4+15+11+11)/5=10.4(s)

平均带权周转时间 W=(11/4+4/2+15/5+11/3+11/3)/5=3.01(s)

当时间片q=4时，调度图如图5.9所示。

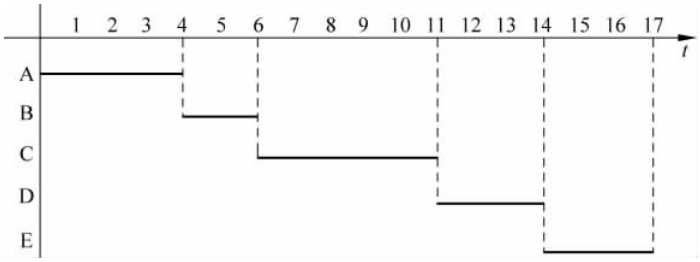


图 5.9　q=4 时的 RR 调度算法运行情况

作业 B 的时间片仅用了 2 个时间单位，剩余时间片可由下一个作业使用。从图 5.9 得出的 5 道作业的完成时间和周转时间如表 5.6 所示。

表 5.6　5 道作业的完成时间和周转时间

作　业　号	进入时间/s	运行时间/s	完成时间/s	周转时间/s
A	0	4	4	4
B	1	2	6	5
C	2	5	11	9
D	3	3	14	11
E	4	3	17	13

平均周转时间 T=(4+5+9+11+13)/5=8.4(s)

平均带权周转时间 W=(4/4+5/2+9/5+11/3+13/3)/5=3.08(s)

可见，在时间片大到一定程度时，RR 调度算法退化成了 FCFS 调度算法。

5.4.4　高优先权优先调度算法

时间片轮转调度算法做了一个内在的假设，即所有的作业或进程同等重要。事实上，每个作业或进程的紧急程度是不相同的。为了使紧急的作业或进程得到优先调度，在许多系统中，每个作业或进程都被指定一个优先级，调度程序总是选择相应队列中具有最高优先权的作业或进程，这就是高优先权优先(First Priority First，FPF)调度算法。这是在批处理系统和实时系统中常用的一种调度算法。该算法的关键是如何确定进程的优先权，常用以下两种方法。

1. 静态优先权

静态优先权是在创建进程时确定的，在整个运行期间不再改变。影响进程静态优先权的主要因素包括进程类型(通常系统进程的优先权高于用户进程的优先权)、进程对资源的要求(如估计的运行时间、内存需要量等)、用户要求的优先级等。

基于静态优先权的调度算法实现简单、系统开销小。但由于静态优先权一旦确定之后，直到运行结束为止始终保持不变，从而使优先级低的进程长时间得不到调度。在现代操作系统中，若使用优先权调度算法，则大多采用动态优先权算法。

2. 动态优先权

动态优先权基于某种原则，使进程的优先权随时间而改变，例如在就绪队列中的进程，其优先权以速度 a 增加，若再令正在执行进程的优先权以速度 b 下降，便可防止一个长作业

长期垄断处理器的情况的发生。

优先数通常为0～4095的整数,是数大的优先权高还是数小的优先权高取决于系统。在UNIX和许多其他的系统中,优先级数值越大,表示的进程优先权越低;而在某些系统中的用法则刚好相反,如Windows系统中,大数值表示高优先级。优先级调度算法通常情况下允许剥夺调度。

【例5-4】 假设有5道作业,它们的进入时间、运行时间和静态优先数如表5.7所示。

表5.7 5道作业的进入时间、运行时间和静态优先数

作 业 号	进入时间/h	运行时间/h	静态优先数
1	0	4	6
2	1	3	4
3	2	6	2
4	3	2	5
5	4	4	3

假定优先数越大,优先级越低,系统采用静态FPF调度算法,分为:

(1) 不可剥夺的高优先权优先调度算法。

(2) 可剥夺的高优先权优先调度算法。

试计算采用该调度算法时的平均周转时间和平均带权周转时间。

扫描二维码,观看视频

解:(1) 不可剥夺的静态FPF调度算法,调度顺序是1、3、5、2、4。

5道作业的执行时间、完成时间和周转时间如表5.8所示。

表5.8 5道作业的执行时间、完成时间和周转时间

作 业 号	进入时间/h	运行时间/h	开始执行时间/h	完成时间/h	周转时间/h
1	0	4	0	4	4
2	1	3	14	17	16
3	2	6	4	10	8
4	3	2	17	19	16
5	4	4	10	14	10

平均周转时间 T=(4+8+10+16+16)/5=10.8(h)

平均带权周转时间 W=(4/4+8/6+10/4+16/3+16/2)/5=3.63(h)

(2) 如果采用可剥夺FPF调度算法,首先进入的是1号作业,当它运行1小时后,2号作业到达,由于已知条件是假定优先数越大,优先级越低,所以2号作业会抢占处理器,2号作业运行1小时后,由于更高优先级的3号作业到达,因此3号作业抢占处理器,而且由于没有其他作业的优先级高于3号作业,所以3号作业一直运行到结束。3号作业运行结束后,由于所有作业均到达,因此按照优先级从高到低运行。注意1号和2号作业后续运行时间加上已占用的运行时间等于所需的运行时间即结束。

5道作业的执行时间、完成时间和周转时间如表5.9所示。

表 5.9　5 道作业的执行时间、完成时间和周转时间

作　业　号	进入时间/h	运行时间/h	开始执行时间/h	占用处理器的过程	完成时间/h	周转时间/h
1	0	4	0	0～1,16～19	19	19
2	1	3	1	1～2,12～14	14	13
3	2	6	2	2～8	8	6
4	3	2	14	14～16	16	13
5	4	4	8	8～12	12	8

平均周转时间 T＝(19＋13＋6＋13＋8)/5＝11.8(h)

平均带权周转时间 W＝(19/4＋13/3＋6/6＋13/2＋8/4)/5＝3.72(h)

5.4.5　最高响应比优先调度算法

最高响应比优先(Highest Response_ratio Next,HRN)调度算法是 FCFS 和 SJF 的一种折中。FCFS 算法只考虑了每个作业的等待时间而未考虑执行时间的长短,而 SJF 算法只考虑了执行时间而未考虑等待时间的长短。因此,这两种调度算法在某些极端情况下会带来某些不便。HRN 调度算法同时考虑每个作业的等待时间和估计的需要执行时间的长短。设响应比为 R,则一种常用的响应比的定义如下:

$$R_p = \frac{W + T}{T} = 1 + \frac{W}{T}$$

其中,W 为等待的时间;T 为该作业估计要执行的时间。

最高响应比优先调度算法的思想是在当前进程完成或被阻塞时,选择 R_p 值最大的就绪进程。这个方法非常具有吸引力,因为它说明了进程的年龄,当偏爱短进程时(因为小分母产生大比率值),长进程由于得不到服务而等待的时间增加,从而 W/T 增加,R_p 也就随之增加,最终在竞争中获胜。

【例 5-5】　假设有 5 道作业,它们的进入时间、运行时间如表 5.10 所示。

表 5.10　5 道作业的进入时间、运行时间

作　业　号	进入时间/h	运行时间/h
1	0	6
2	1	4
3	2	4
4	3	1
5	4	3

系统采用 HRN 调度算法,试计算采用该调度算法时的平均周转时间和平均带权周转时间。

解:采用 HRN 调度算法,应在每次调度时计算各作业的响应比,选择响应比高的作业进入运行状态。

在时刻 0,由于只有作业 1,因此调度作业 1 投入运行,作业 1 运行到时刻 6 结束,系统需在时刻 6 进行调度;

在时刻 6,计算各作业的响应比为:

扫描二维码,观看视频

作业2的响应比：$R_2=1+(6-1)/4=2.25$

作业3的响应比：$R_3=1+(6-2)/4=2$

作业4的响应比：$R_4=1+(6-3)/1=4$

作业5的响应比：$R_5=1+(6-4)/3=1.67$

选择作业4投入运行，作业4运行到时刻7时结束，在时刻7时进行调度；

在时刻7，计算各作业的响应比为：

作业2的响应比：$R_2=1+(7-1)/4=2.5$

作业3的响应比：$R_3=1+(7-2)/4=2.25$

作业5的响应比：$R_5=1+(7-4)/3=2$

选择作业2投入运行，作业2运行到时刻11时结束，在时刻11时进行调度；

在时刻11，计算各作业的响应比为：

作业3的响应比：$R_3=1+(11-2)/4=3.25$

作业5的响应比：$R_5=1+(11-4)/3=3.33$

选择作业5投入运行，作业5运行到时刻14时结束，在时刻14时进行调度；

在时刻14，只有作业3等待运行，因此5道作业的调度顺序是1、4、2、5、3。

5道作业的运行时间、完成时间和周转时间如表5.11所示。

表5.11　5道作业的执行时间、完成时间和周转时间

作　业　号	进入时间/h	运行时间/h	开始执行时间/h	完成时间/h	周转时间/h
1	0	6	0	6	6
2	1	4	7	11	10
3	2	4	14	18	16
4	3	1	6	7	4
5	4	3	11	14	10

平均周转时间 $T=(6+4+10+10+16)/5=9.2$(h)

平均带权周转时间 $W=(6/6+4/1+10/4+10/3+16/4)/5=2.97$(h)

5.4.6　多级队列调度算法

多级队列调度算法(Multiple-level Queue，MQ)的基本思想是引入多个就绪队列，通过对各队列的区别对待，达到一个综合的调度目标。在多级队列调度算法中，根据作业或进程的性质或类型的不同，将就绪队列再分成若干个子队列，每个作业固定归入一个队列，例如，系统进程、用户交互进程、批处理进程等不同队列。不同队列可有不同的优先级、时间片长度、调度策略等。

5.4.7　多级反馈队列调度算法

多级反馈队列(Round Robin with Multiple Feedback，RRMF)调度算法是时间片轮转算法和优先级调度算法的综合和发展。通过动态调整进程优先级和时间片大小，多级反馈队列算法可兼顾多方面的系统目标。例如，为提高系统吞吐量和缩短平均周转时间而照顾短进程；为获得较好的I/O设备利用率和缩短响应时间而照顾I/O型进程；同时，也不必

事先估计进程的执行时间。

在多级反馈队列调度算法中，通常设置多个就绪队列，并赋予各队列不同的优先级。如队列 1 的优先级最高，然后逐级降低。每个队列的执行时间片长度也不同，如规定优先级越低则时间片越长。新进程进入内存后，先投入最高优先级队列 1 的末尾，按 FCFS 调度算法调度；如果在队列 1 的一个时间片内未能执行完，则降低投入到队列 2 的末尾，同样按 FCFS 调度算法调度；如此下去，一直降低到最后的队列，则按时间片轮转法调度直到完成。仅当较高优先级的队列为空时，才调度较低优先级队列中的进程运行。如果进程运行时有新进程进入较高优先级队列，则抢先执行新进程，并把被抢先的进程投入原队列的末尾。多级反馈队列调度算法如图 5.10 所示。

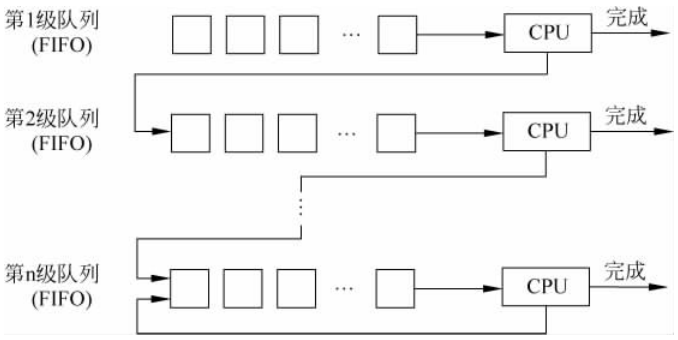


图 5.10　多级反馈队列

在实际系统中，使用多级反馈队列调度算法还可以采用更复杂的动态优先级调整策略。例如，为了保证 I/O 操作的及时完成，通常会在进程发出 I/O 请求后进入最高优先级队列，并运行一个较小的时间片，以及时响应 I/O 交互。对于计算型进程，可在每次执行完一个完整的时间片后，进入更低级队列，并最终采用最大时间片来执行，这样可减少计算型进程的调度次数。对于 I/O 次数不多而处理器占用时间较多的进程，可使用高优先级进行 I/O 处理，在 I/O 完成后，放回原来队列，以免每次都从最高优先级队列逐次下降。一种更通用的策略是进行 I/O 时提高优先级；时间片用完时降低优先级。这样可适应一个进程在不同时间段的运行特点。

5.5　实 时 调 度

在实时系统中，可能存在着两类不同性质的实时任务，即硬实时(Hard Real Time，HRT)任务和软实时(Soft Real Time，SRT)任务，它们都联系着一个截止时间。为保证系统能正常工作，实时调度必须能满足实时任务对截止时间的要求。为此，实现实时调度应具备一定的条件。

5.5.1　实现实时调度的基本条件

1. 提供必要的信息

为了实现实时调度，系统应向调度程序提供有关任务的信息：

(1) 就绪时间，是指某任务成为就绪状态的起始时间，在周期任务的情况下，它是事先

预知的一串时间序列。

(2) 开始截止时间和完成截止时间,对于典型的实时应用,只需知道开始截止时间,或者完成截止时间。

(3) 处理时间,一个任务从开始执行,直至完成时所需的时间。

(4) 资源要求,任务执行时所需的一组资源。

(5) 优先级,如果某任务的开始截止时间错过,势必引起故障,则应为该任务赋予"绝对"优先级;如果其开始截止时间的错过,对任务的继续运行无重大影响,则可为其赋予"相对"优先级,供调度程序参考。

2. 系统处理能力强

在实时系统中,若处理机的处理能力不够强,则有可能因处理机忙不过来,而致使某些实时任务不能得到及时处理,从而导致发生难以预料的后果。假定系统中有 m 个周期性的硬实时任务 HRT,它们的处理时间可表示为 C_i,周期时间表示为 P_i,则在单处理机情况下,必须满足下面的限制条件系统才是可调度的:

$$\sum_{i=1}^{m}\frac{C_i}{P_i}\leqslant 1$$

例如,有 5 个周期性硬实时任务,每个任务的周期时间都是 40ms,每个任务的处理时间为 10ms,则

$$\sum_{i=1}^{m}\frac{C_i}{P_i}=\sum_{i=1}^{6}\frac{C_i}{P_i}=\frac{10}{40}\times 5=\frac{5}{4}>1$$

则该系统是不可调度的。

这个限制条件并未考虑到任务切换所花费的时间,因此,当利用上述限制条件时,还应适当考虑留有余地。提高系统处理能力的途径有二:一是采用单处理机系统,但需增强其处理能力,以显著地减少对每一个任务的处理时间;二是采用多处理机系统。假定系统中的处理机数为 N,则应将上述的限制条件改为:

$$\sum_{i=1}^{m}\frac{C_i}{P_i}\leqslant N$$

3. 采用抢占式调度机制

在含有 HRT 任务的实时系统中,广泛采用抢占机制。这样便可满足 HRT 任务对截止时间的要求。但这种调度机制比较复杂。

4. 具有快速切换机制

为保证硬实时任务能及时运行,在系统中还应具有快速切换机制,使之能进行任务的快速切换。该机制应具有如下两方面的能力:

(1) 对中断的快速响应能力。对紧迫的外部事件请求中断能及时响应,要求系统具有快速硬件中断机构,还应使禁止中断的时间间隔尽量短,以免耽误时机(其他紧迫任务)。

(2) 快速的任务分派能力。为了提高分派程序进行任务切换时的速度,应使系统中的每个运行功能单位适当小,以减少任务切换的时间开销。

5.5.2 实时调度算法的分类

可以按不同方式对实时调度算法加以分类:

(1) 根据实时任务性质,可将实时调度的算法分为硬实时调度算法和软实时调度算法;

(2) 按调度方式,则可分为非抢占调度算法和抢占调度算法。

1. 非抢占式调度算法

(1) 非抢占式轮转调度算法:进程排成一个轮转队列,调度程序每次选择队首的进程运行。当实时进程提出请求时,也要排队等候,轮到该实时进程时才被调度运行。这种算法可用于要求不太严格的实时控制系统中。

(2) 非抢占式优先调度算法:如果实时任务被赋予了较高的优先级,可以将这些实时进程安排在就绪队列的队首,等待到当前进程终止或运行完成后,就可以调度排在队首的高优先级实时进程。这种算法需要精心安排任务的优先级,可用于有一定要求的实时系统中。

2. 抢占式调度算法

可根据抢占发生时间的不同而进一步分成以下两种调度算法。

(1) 基于时钟中断的抢占式优先级调度算法:如果某实时任务到达,而且它的优先级高于当前进程的优先级,此时不能立即抢占处理机,而是等到时钟中断发生时,调度程序才剥夺当前进程的执行,将处理机分配给新到的高优先级实时任务,这种算法可用于大多数的实时系统中。

(2) 立即抢占(immediate preemption)的优先级调度算法:这种调度策略是一旦出现高优先级实时进程,只要当前进程不在临界区中,就立即剥夺当前进程的执行,将处理机分配给新到的高优先级实时任务。这种算法是响应时间最短的。

图 5.11 中的(a)、(b)、(c)、(d)分别给出了 4 种情况的调度时间。

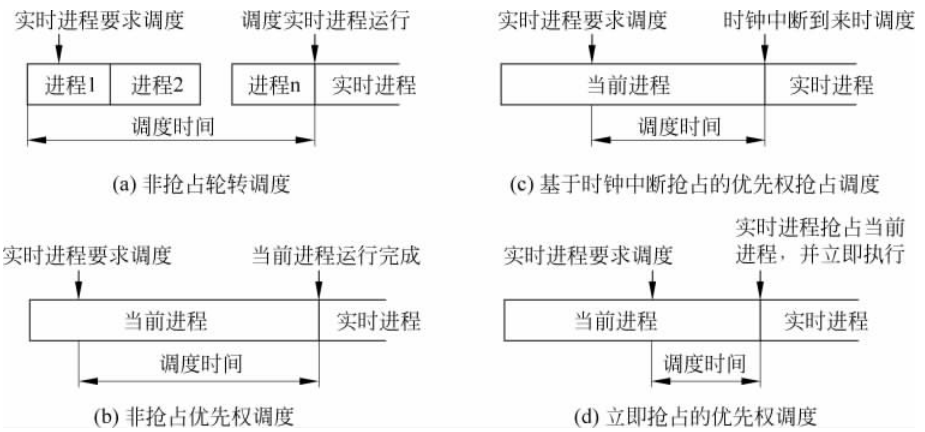


图 5.11 实时进程调度

5.5.3 最早截止时间优先算法

最早截止时间优先(Earliest Deadline First,EDF)算法是根据任务的截止时间来确定任务的优先级。任务的截止时间越早,其优先级越高。具有最早截止时间的任务排在队首。由于就绪队列中任务按其截止时间排列,因此队首任务先分配处理机。

非抢占式调度方式用于非周期实时任务,抢占式调度方式用于周期实时任务。

1. 非抢占式调度方式用于非周期实时任务

图 5.12 给出了将 EDF 算法用于非抢占调度方式的例子。图中有 4 个非周期性任务,

它们的到达次序如图中下方所示。由于任务1最先到达,因此系统首先调度任务1运行,在任务1运行期间,任务2和任务3先后到达。由于任务3的开始截止时间早于任务2的,所以系统在任务1完成后先调度任务3执行。在任务3运行期间任务4又到达,由于任务4的开始截止时间早于任务2的,因此在任务3执行完毕后,系统调度任务4执行,最后调度任务2。

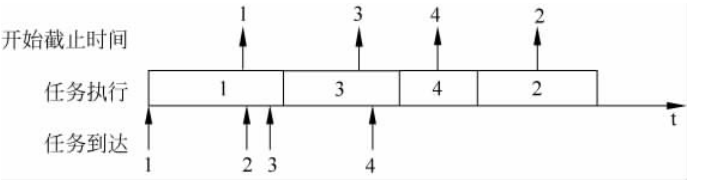


图5.12 EDF算法用于非抢占调度方式

2. 抢占式调度方式用于周期实时任务

图5.13示出了将该算法用于抢占调度方式之例。在该例中有两个周期性任务,任务A的周期时间为20ms,每个周期处理时间为10ms;任务B的周期时间为50ms,每个周期处理时间为25ms。两个任务的到达时间,最后期限(也就是截止时间)和执行时间如图5.13所示。

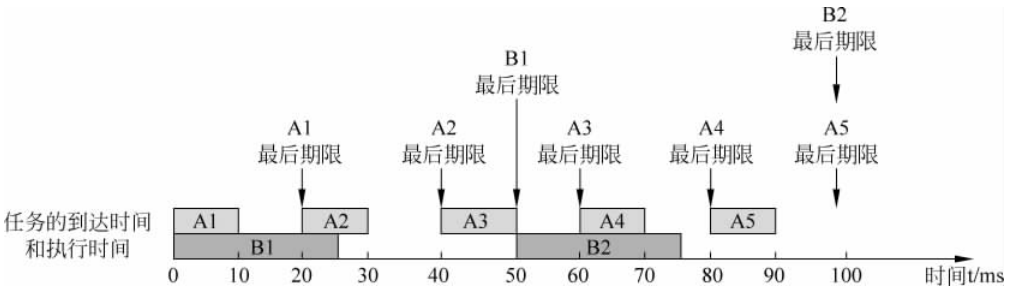


图5.13 最早截止时间优先算法的要求示例

图5.13中的条形方块显示了两个周期性任务A和B的到达时间、截止时间和执行时间示意图。其中任务A的到达时间为0、20ms、40ms……任务A的最后期限为20ms、40ms、60ms……任务B的到达时间为0、50ms、100ms……任务B的最后期限为50ms、100ms、150ms……从图5.13中可以明显看出,任务A和B的执行时间的需求是有冲突的,那么应该如何合理安排这两个周期性任务呢?

在这样的周期性任务中,如果采用通常的优先级调度算法,是无法保证任务的正常执行的。请参看图5.14,假定任务A具有较高的优先级,那么在t=0ms时,先调度A1执行,A1完成后(t=10ms时)调度B1执行。在t=20ms时,由于高优先级的任务A2已到达,所以B1被抢占,优先执行A2。A2完成后(t=30ms时),继续执行被中断的B1,可是10ms后,即t=40ms时,高优先级的任务A3又到达,于是B1再次被抢占,优先执行A3。A3完成后(t=50ms时),B1尚未执行完毕,但已错过了它的最后期限,说明通常的优先级调度算法在这里是失败的。

类似地,如果假定任务B具有较高的优先级,参看图5.15,那么在t=0ms时,先调度

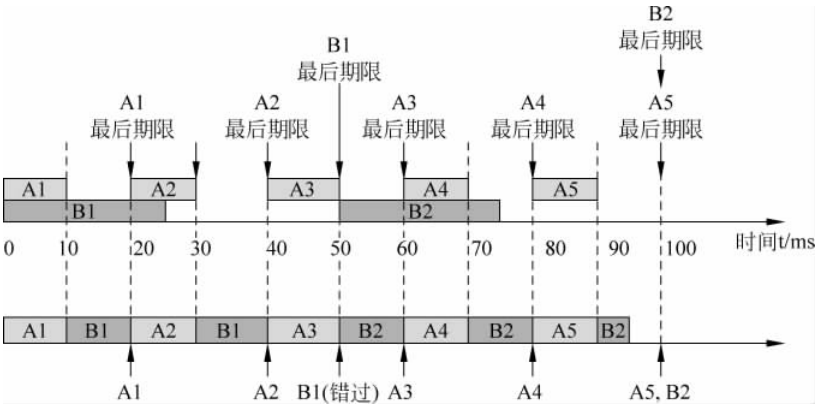


图 5.14　任务 A 具有较高优先级时通常的调度算法调度失败示意图

B1 执行，B1 完成后(t＝25ms 时)，任务 A1 的最后期限已错过。接下来只能执行任务 A2 和任务 A3，当 t＝50ms 时，由于任务 B 具有较高的优先级，因此选择执行 B2，又会再次导致任务 A4 错过最后期限。因此通常的优先级调度算法在这里还是行不通的。

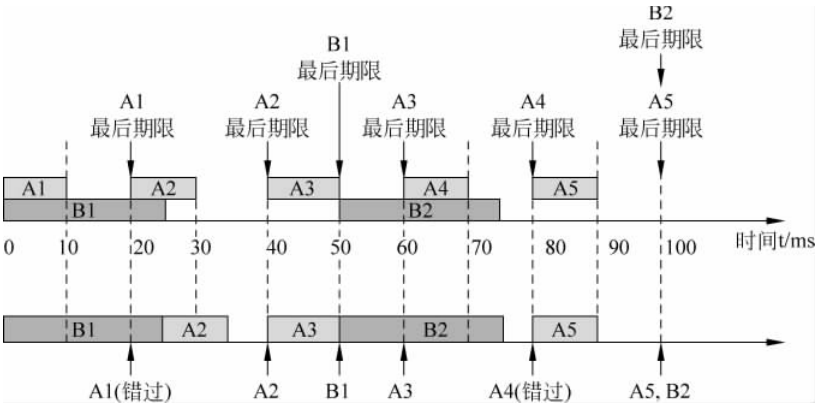


图 5.15　任务 B 具有较高优先级时通常的调度算法调度失败示意图

综上所述，只能采用可抢占的最早截止时间优先调度算法来完成周期性实时任务。如图 5.16 所示，在 t＝0 时，A1 和 B1 同时到达，由于 A1 的截止时间比 B1 早，因此先调度 A1 执行。A1 完成后(t＝10ms 时)调度 B1 执行。在 t＝20ms 时，A2 到达，由于 A2 的截止时间比 B1 早，因此 B1 被中断调度 A2 执行。A2 完成后(t＝30ms 时)重新调度 B1 继续执行。在 t＝40ms 时，A3 到达，但是由于 B1 的截止时间要比 A3 早，因此仍然让 B1 继续执行直到完成(t＝45ms)，然后在调度 A3 执行。当 t＝55ms 时，A3 完成又调度 B2 执行。从图 5.16 中可以看出，利用谁的截止时间更早，谁就优先执行的算法思想，可以满足系统的要求，不会错过任务的最后期限。

5.5.4　最低松弛度优先算法

最低松弛度优先(Least Laxity First，LLF)算法在确定任务的优先级时，根据的是任务的紧急(或松弛)程度。任务紧急程度愈高，赋予该任务的优先级就愈高，以使之优先执行。

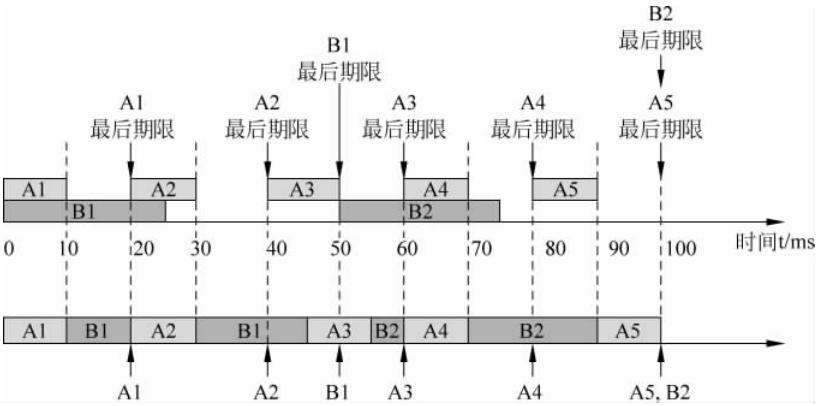


图 5.16　最早截止时间优先算法用于抢占调度方式示例

例如有一个任务最迟在 200ms 时必须完成，而它本身所需的运行时间是 50ms，那么该任务可以最早在 0～50ms 的时间段内执行，也可以最晚在 150～200ms 时间段执行完毕即可，调度程序最晚必须在 150ms 时调度它执行，则该任务的紧急程度(松弛程度)为 150ms。使用松弛度这个概念就是因为任务有一定的缓冲区间，就像橡皮筋一样可以绷紧，也可以放松。松弛度越小说明任务越紧急，越需要马上完成。使用该算法在系统中要建立一个按照松弛程度排序的任务队列，松弛度最低的排在最前面，松弛度越大的排在越后面，调度程序选择队列的队首任务运行。

该算法主要用于可抢占调度方式中。还是与上面一样的例子，假如在一个实时系统中有两个周期性实时任务 A 和 B，任务 A 要求每 20ms 执行一次，执行时间为 10ms，任务 B 要求每 50ms 执行一次，执行时间为 25ms。由此可知，任务 A 和 B 每次最迟必须完成的时间分别为 A_1(20ms)、A_2(40ms)、A_3(60ms)……和 B_1(50ms)、B_2(100ms)、B_3(150ms)……如图 5.17 所示。

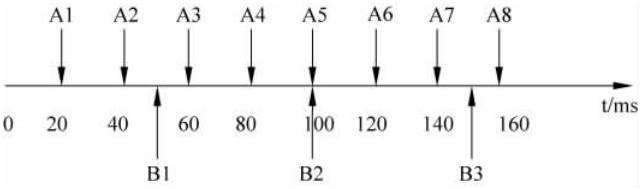


图 5.17　A 和 B 任务每次最迟必须完成的时间

给出计算松弛度的公式如下：

松弛度＝必须完成的时刻点－任务剩余需要的运行时间－当前时间

在刚开始时(t＝0 时)，由于 A1 最晚必须在 20ms 时完成，而它本身运行又需要 10ms，因此可以算出 A1 的松弛度＝20－10－0＝10ms。而 B1 必须在 50ms 时完成，它本身的运行时间需要 25ms，可算出 B1 的松弛度＝50－25－0＝25ms，根据最低松弛度优先算法，故调度程序优先调度 A1 运行。当 A1 运行完成(t＝10ms 时)，B1 的松弛度＝50－25－10＝15ms，而 A2 的松弛度＝40－10－10＝20ms，故调度程序选择 B1 优先运行。注意：当 t＝30ms 时，由于 A2 的松弛度已减少为 40－10－30＝0ms，而此时 B1 的松弛度＝50－5－

30＝15ms，因此调度程序要立即抢占 B1 的处理机让 A2 运行。A2 运行完毕(t＝40ms 时)，A3 的松弛度＝60－10－40＝10ms，而 B1 的松弛度仅为 50－5－40＝5ms，所以应重新调度 B1 执行。B1 完成完毕(t＝45ms 时)，A3 的松弛度＝60－10－45＝5ms，而 B2 的松弛度＝100－25－45＝30ms，所以优先选择 A3 运行。类似地，读者可以分析得具有两个周期性实时任务的调度情况，如图 5.18 所示。可以观察发现，由于周期性任务会有规律地反复发生，所以 100ms 后面不需要再作图，只重复前面的规律即可。

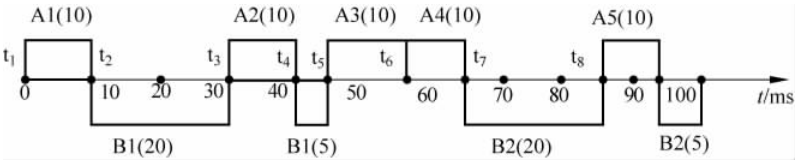


图 5.18　利用 ELLF 算法进行调度的情况

5.6　实例分析：UNIX 进程调度

UNIX System V 的进程调度涉及调度时机、调度标记设置、优先数计算、调度的实现等。下面将分别进行说明。

5.6.1　调度时机

UNIX System V 在以下 5 种情况之一发生时进行进程调度：

(1) 现行进程自己调用 sleep 或 wait 等进入睡眠状态时。

(2) 现行进程调用 exit 自我终止时。

(3) 现行进程的时间片到期且优先级低于其他就绪进程时。

(4) 现行进程在完成中断和陷入处理后返回用户态时，其优先级已低于其他就绪进程或者调度标记被置位。

(5) 现行进程从系统调用执行结束后返回用户态时，其优先级已低于其他就绪进程或者调度标记被置位。

上述(1)、(2)、(3)中的情况是容易理解的，(4)、(5)中的情况是 UNIX 设计者采用的独特技术。显然，进程调度应在核心态中完成，但在核心态下随机发生进程调度将使 UNIX 系统的设计复杂化。于是 UNIX 设计者把核心态下进程调度的时机都集中在系统程序运行结束后返回用户态之前的瞬间，保证在系统程序执行过程中不发生进程调度或能预知何时发生进程调度。这虽然牺牲了进程并发程度但却简化了操作系统设计中进程的同步与可重入码问题。人们把 UNIX 核心态下进程的运行方式称为伪异步方式。

5.6.2　调度标记设置

runrun 是要求进行进程调度时的标记。若 wakeup、setrun 以及 setpri(设置优先级)命令发现某进程的优先级高于现行进程，则置 runrun 为 1。另外在秒中断处理过程中也将检查各就绪进程的优先级，发现优先级高于现行进程时，同样置 runrun 为 1。

5.6.3 优先数计算

传统的UNIX System V采用动态优先数调度策略,优先数越大优先级越低。调度程序从内存就绪队列中选取优先数最小的进程作为上行进程。优先数基于进程类型和运行历史,计算公式如下:

$$p\_pri=\frac{p\_CPU}{2}+PUSER+p\_nice+NZERO$$

其中,PUSER和NZERO分别取常数25和20,为基本用户优先数的阈值;p_CPU为进程最近一次使用CPU的时间。当进程使用CPU时,系统每个时钟周期对该进程的p_CPU加1,该值最大可达80。此外,秒中断时对p_CPU执行除以2的衰减操作。这样,如果系统的时钟周期为16.667ms,则遇秒中断时p_CPU将由60变为30;p_nice是系统允许用户使用nice系统调用影响进程优先数的偏移值,范围在0～40,一旦设置后,普通用户只能做递增修改。

由于新创建的进程未使用CPU,其p_CPU的值为0,因而具有较高的优先级。但随着被调度执行,其p_CPU的值将会不断增加,在秒中断处理时其CPU有可能被其他高优先级的进程所抢夺。但随着每秒1次的p_CPU值的衰减,其优先级将会提高,总有机会能重新夺回处理器。

5.6.4 调度的实现

UNIX系统的进程调度是由switch过程实现的,switch过程是进程0的一部分。

首先,调度程序把现行进程的上下文(主要是寄存器上下文和栈指针)保存到核心栈或user结构内的u_rsav、u_pcb字段中,这是由过程save(u.u_rsav)完成的。

其次,按计算优先数的公式从内存就绪队列中寻找优先级最高的进程作为上行进程。若内存就绪队列中不存在这样的进程,则调用空闲进程等待某进程变成就绪状态。清除runrun标记。

最后,调用resume过程恢复上行进程的上下文,恢复核心栈或user结构内的u_rsav及u_pcb,从而使上行进程变成现行进程。

由于内存空间的限制,UNIX进程的上下文有时存放在外存。因此,进程调度选中某一进程作为上行进程时,必须把其处于外存的上下文调进内存。如果此时没有足够的内存能容纳这个上下文,则要设法调出某些进程到外存,这就是所谓的对换过程,由Sched过程完成,Sched过程也是进程0的一部分。

本章小结

作业与进程的调度是操作系统对处理器进行管理的具体体现。通过本章的学习,掌握三级调度的概念、作业调度的功能及其目标与性能衡量、进程调度的功能及方法,重点掌握作业与进程的调度算法,并能利用这些算法解决实际问题。

作业具有4种状态:提交状态、后备状态、运行状态和完成状态。了解4种状态的含义及其转换条件和时机。

三级调度指的是：高级调度，即作业调度；中级调度，即交换调度；低级调度，即进程调度。

作业调度的目标与性能衡量是设计操作系统时必须遵循的准则，但用户和系统衡量系统好坏的标准是不一致的，因此分为面向系统的准则和面向用户的准则两类。

作业与进程的调度算法有多种。要学会根据系统性能及目标的不同，选择相应的调度算法。能够利用各种算法计算作业的平均周转时间和平均带权周转时间。

习　　题

1. 单项选择题

(1) 当作业进入完成状态时，操作系统(　　)。

A. 将删除该作业并收回其所占资源，同时输出结果

B. 将该作业的控制块从当前作业队列中删除，收回其所占资源，并输出结果

C. 将收回该作业所占资源并输出结果

D. 将输出结果并删除内存中的作业

(2) 某系统正在执行 3 个进程 P1、P2 和 P3，各进程的计算(占用 CPU)时间和 I/O 时间比例如表 5.12 所示，为提高系统资源利用率，合理的进程优先设置为(　　)。

表 5.12　进程的计算时间和 I/O 时间

进　　程	计 算 时 间	I/O 时间
P1	90%	10%
P2	50%	50%
P3	15%	85%

A. P1＞P2＞P3　　B. P3＞P2＞P1　　C. P2＞P1＝P3　　D. P1＞P2＝P3

(3) 下列进程调度算法中，综合考虑进程等待时间和执行时间的是(　　)。

A. 最高响应比优先调度算法　　B. 先来先服务调度算法

C. 短进程优先调度算法　　D. 时间片轮转调度算法

(4) 在批处理系统中，周转时间是(　　)。

A. 作业运行时间

B. 作业等待时间和运行时间之和

C. 作业的等待时间

D. 作业被调度进入内存到开始运行的时间

(5) 在操作系统中，作业处于(　　)状态时，已处于进程管理之下。

A. 提交　　B. 后备　　C. 运行　　D. 完成

(6) 下列选项中，降低进程优先级的合理时机是(　　)。

A. 进程的时间片用完　　B. 进程刚完成 I/O，进入就绪队列

C. 进程长期处于就绪队列中　　D. 进程从就绪状态转为运行状态

(7) 一个作业被成功调度后，系统为其创建相应的进程，该进程的初始状态是(　　)。

A. 执行态　　B. 阻塞态

C. 就绪态　　D. 等待访问设备态

(8) 下列选项中,满足短作业优先而且不会发生饥饿现象的调度算法是(　　)。

A. 先来先服务　　B. 最高响应比优先

C. 时间片轮转　　D. 非抢占式短任务优先

(9) 若某单处理器多进程系统中有多个就绪态进程,则下列关于处理机调度的叙述中,错误的是(　　)。

A. 在进程结束时能进行处理机调度

B. 创建新进程后能进行处理机调度

C. 在进程处于临界区时不能进行处理机调度

D. 在系统调用完成并返回用户态时能进行处理机调度

2. 填空题

(1) 作业调度是从处于(　　)状态的队列中选取适当的作业投入运行。从作业提交给系统到作业完成的时间间隔称为(　　)。(　　)是指作业从进入后备队列到被调到程序中的时间间隔。假定把如表5.13所示的4个作业同时提交系统并进入(　　)队列,当使用短作业优先调度算法时,单道环境下,4个作业的平均等待时间是(　　),平均周转时间是(　　);当使用高优先数优先的调度算法时,作业的平均等待时间是(　　),平均周转时间是(　　)。

表5.13　4个作业的运行时间和优先数

作　业	所需运行时间/h	优　先　数
1	2	4
2	5	9
3	8	1
4	3	7

(2) 在一个具有分时兼批处理的系统中,总是优先调度(　　)。

3. 简答题

(1) 什么是分层次调度?在分时系统中有作业调度的概念吗?如果没有,为什么?

(2) 作业调度和进程调度的主要功能分别是什么?

(3) 作业调度的性能评价标准有哪些?这些性能评价标准在任何情况下都能反映调度策略的优劣吗?

(4) 为什么说多级反馈队列调度算法能较好地满足各类用户的需要?

(5) 假设就绪队列中有10个进程,系统将时间片设为200ms,CPU进行进程切换要花费10ms,试问系统开销所占的比率约为多少?

(6) 在批处理系统、分时系统和实时系统中一般常采用哪种调度算法?

(7) 若在后备作业队列中等待运行的同时有3个作业1、2、3,已知它们各自的运行时间为a、b、c,且满足关系 a<b<c,试证明采用短作业优先调度算法能获得最小的平均周转时间。

4. 应用题

(1) 考虑5个进程 P_1、P_2、P_3、P_4、P_5,它们的创建时间、运行时间及优先数如表5.14所示。规定进程的优先数越小,优先级越高。试描述在采用下述几种调度算法时各个进程的

运行过程，并计算采用每种算法时的进程平均周转时间。假设忽略进程的调度时间。

表 5.14　5 个进程的创建时间、运行时间和优先数

进　　程	创 建 时 间	运行时间/ms	优　先　数
P_1	0	3	3
P_2	2	6	5
P_3	4	4	1
P_4	6	5	2
P_5	8	2	4

① 先来先服务调度算法。

② 短进程优先调度算法。

③ 时间片轮转调度算法(时间片为 1ms)。

④ 非剥夺式优先级调度算法。

⑤ 剥夺式优先级调度算法。

⑥ 最高响应比优先调度算法。

(2) 有一个具有两道作业的批处理系统，作业调度采用短作业优先的调度算法，进程调度采用以优先数为基础的剥夺式调度算法。表 5.15 所示为作业序列，作业优先数即为进程优先数，优先数越小优先级越高。

表 5.15　两道作业的运行时间和优先数

作　　业	到 达 时 间	估计运行时间/min	优　先　数
A	10:00	40	5
B	10:20	30	3
C	10:30	50	4
D	10:50	20	6

① 列出所有作业进入内存的时间及结束时间。

② 计算平均周转时间。