第3章

进程与处理器管理

处理器是计算机系统执行程序的重要器件，操作系统要管理好处理器，提高处理器的利用率，必须能够在一个程序因为启动了 I/O 而需要等待 I/O 数据时，让另一个不相关的程序能够被调度到处理器上运行。程序没有执行完时，如果要让出处理器，操作系统必须对它们的运行状态进行记录，确保不同程序能够**分时**占用处理器并正确运行。

为了实现上述目的，操作系统要用适当的表格来表示程序的执行，当然运行之前要分配存放程序和数据的主存等系统资源，保证用户程序能够顺利地在处理器上运行。

应该说，组织用户使用计算机的机制是随着计算机操作系统的发展而进化的。在监督程序时代是以作业形式表示程序运行的，那时，作业以同步方式串行地运行每个作业必须完成的步骤，如串行地编译、链接、运行目标程序；当操作系统发展到分时系统时，为了开发同一道作业中不同作业步之间的并发，作业机制已不能满足需要，因而引入了**进程**机制，让进程来实现作业步的执行。但随着多处理器计算机的出现，用户希望一个作业步中的程序还能够同时在多个处理器上运行，因此进程的机制得到了进一步发展，即让一个进程同时拥有多个**线程**，让一个进程的多个线程在不同处理器上同时运行。

这里假设系统只有一个处理器，而如何把处理器合理有效地分配给各执行程序使用是处理器管理的主要内容。

本章将讨论如何在计算机系统中表示程序及其执行；如何管理和控制程序的执行；如何组织和协调程序对处理器的争夺使用，最大限度地提高处理器的利用率。

本章主要内容如下。

(1) 进程管理。进程是操作系统的一个重要概念，进程是从批处理系统中为了支持作业及作业步并发而发展出来的机制。本章将重点介绍进程的内部表示、进程的状态及变化、进程相关系统调用及调度与切换等。

(2) 进程与作业的关系。在早期监督程序时代，作业是处理器使用单位，那时没有进程概念，在提出了多道程序设计思想及分时系统出现后，进程概念被引入。本章将介绍在引入进程概念以后作业与进程的关系。

(3) 线程引入。引入线程的目的是为了支持进程中程序的并发/并行执行。原来的进程只有一个占用处理器的执行单位,在引入线程概念后,一个进程内可以有多个线程,每个线程都可以被调度占用处理器。这样,线程之间完全可以并发或并行执行,而且它们还都共享进程的程序和数据空间。

3.1　进 程 描 述

为什么会引入进程的概念? 主要目的就是要实现多个程序在处理器上的轮流运行,以进程为单位来使用计算机的各种资源,包括程序和数据所用的空间、系统外部设备、文件等资源,特别是以分时共享的方式使用处理器资源。可以理解成一个进程就是一个处理器的实例,是虚拟化的处理器,程序在进程上运行。

操作系统的进程与处理器管理模块负责创建进程、为进程加载用户态运行程序、调度进程占用处理器、支持进程间通信等。而其他的资源管理模块负责其他如主存、外设、文件资源的管理。可以把操作系统看成支持进程并且对进程所用系统资源进行管理的系统。

图 3.1 可以解释这一概念。在多道程序设计环境中,有 n 个进程(P_1,P_2,…,P_n)被创建。操作系统定义了各自独立的进程用户空间,用户空间存放用户态运行的程序及处理数据,每个进程在其执行过程中需要访问某些系统资源,包括处理器、物理主存、I/O 设备、文件等。如图 3.1 所示,进程 P_1 正在运行,该进程占有若干物理主存用于程序和数据,并且控制了两个 I/O 设备。进程 P_2 也占有若干物理主存,但由于申请的 I/O 设备已经因 P_1 独占而被阻塞,进入 I/O 设备的等待队列,进程 P_n 正在等待分配物理主存。下面要弄清楚,操作系统如何表示一个进程,如何控制进程及管理进程所需用的资源。

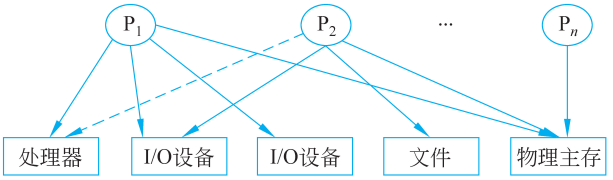

图 3.1　进程与资源

3.1.1　进程定义

进程是支持程序执行的机制。进程可以理解为一个正在运行的虚拟处理器,也可以理解为程序对数据或请求的处理过程。

1. 进程组成

进程由以下 4 方面组成。

(1) 进程包括至少一个可执行程序,含有代码和初始数据,一般在进程创建时说明。注意,可执行程序可以被多个进程共享,换句话说,多个进程可能运行同一个可执行程序。

(2) 进程包括一个独立的进程用户空间,在进程创建时由操作系统分配。

(3) 进程使用的其他系统资源。这是指在进程创建及执行过程中,由操作系统分配给进程的系统资源,包括 I/O 设备、文件等。

(4) 进程包括一个执行栈区,包含程序运行现场信息,如子程序调用时所压栈帧、系统调用时所压栈帧等,这是进程运行及进程调度进行处理器切换时所要涉及的数据结构。

用户空间用来存放用户程序和数据。在为进程分配用户空间的同时,操作系统通常还会为用户程序分配运行时所必需的初始系统资源,如程序进行输入/输出所用的标准 I/O 设备(通常是终端),并且在用户空间中加载用户态运行程序和初始数据。在进程运行过程中,操作系统不断地将系统资源以独占方式或者与其他进程共享的方式分配给进程。

在进程创建时,往往会在用户空间定义一个**用户栈**,用来在用户态运行时保存用户程序现场。在操作系统嵌入用户进程运行模式中,系统还会为进程在操作系统内核空间分配一个**核心栈**,用来保存中断/异常点现场,以及在进程运行核心态程序后的转子现场。逻辑上,进程的用户栈和核心栈都属于一个执行栈区。

一个进程至少运行一个可执行程序,程序往往以文件形式存放于辅存中,程序文件中还包含全局变量数据及常数。因此,一个进程需要足够的存储空间来存放进程的程序和数据。为了执行程序,操作系统还必须为进程分配一个栈区,用来保存子程序调用时的现场或在栈中分配局部变量空间。

如果进程要执行多个程序文件中的程序,操作系统则提供相应的系统调用来支持新文件程序和数据对老程序与数据存储空间的覆盖(如 POSIX 标准的 exec 系列系统调用)。

同一个程序可以由多个进程分别执行,当然,不同的进程虽然执行的是相同的程序,但是处理不同的数据,这种程序被称为共享程序。编制共享程序的技术是研制软件(包括操作系统内核)的重要技术。可共享的程序必须是**可再入(Re-entry)代码**。任何一个程序,逻辑上都可以将其分为两部分:执行过程中不改变自身的不变部分和变量等可变部分。程序内的指令、常量本身不会因程序的执行而发生变化。而程序中的数据区、变量存储区的信息将随着程序的执行发生不同的变化。

为了使程序能成为可再入代码,有效的方法是将其中的可变部分作为与进程相关的环境信息与不变部分的程序分开。如函数内定义的变量在“运行栈”上存储。由于每个进程均有自己的“运行栈”,因此不会发生不同进程运行相同程序情况下的中间结果相互覆盖。当然程序定义的全局变量存放于数据区,但是只要数据区与程序区分开,每个进程拥有自己独立于其他进程的数据区,同一程序被不同进程运行就不会有问题。

2. 进程映像

我们把程序、数据、用户栈的集合称作**进程映像**(Process Image)。

进程映像放在进程的用户空间,如何放于主存取决于存储管理机制。在实存系统中,进程运行时进程映像都已经存放在主存中,现代操作系统几乎都采用页式虚存管理机制,操作系统为进程定义了独立的用户虚空间,在用户虚空间存放程序、数据和用户栈。在进

程创建时会分配并初始化进程虚空间，进程执行新的程序时也会用新的进程映像重新初始化进程用户虚空间。

初始化进程空间是指将辅助存储器中的可执行程序文件中的程序加载到进程空间，并依照可执行程序文件中全局变量的数据说明，分配进程的数据区空间并对其初始化，还要分配好栈区。

3.1.2　进程控制块

操作系统要管理进程和它使用的资源，必须拥有每个进程和资源的描述信息及当前状态信息。操作系统建立和维护表格来表示这些信息，信息包含进程和资源的标识、描述和状态。信息不一定要存放在一片连续空间，可以通过表格中的指针将有关信息连接起来，通过这些指针，操作系统很容易得到下一个要处理的信息。

操作系统管理和控制一个进程需要什么信息呢？操作系统必须建立一个表格用来描述该进程的存在及状态。这个表格被称为**进程控制块**(Process Control Block，PCB)。它描述进程标识、空间、运行状态、资源使用等信息。

操作系统管理着大量的进程，每个进程的管理信息被存放于各自的进程控制块中。各操作系统的实现方式不同，信息的组织方法也不一样。下面先介绍操作系统管理进程用到的数据。

如图 3.2 所示，进程控制块包含下述三大类信息。

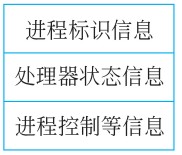

图 3.2　进程控制块(PCB)

1. 进程标识信息

在进程控制块中存放的标识信息主要有本进程的唯一标识、本进程的产生者标识(父进程标识)、进程所属的用户标识。系统每个用户都有可能产生多个进程，用户标识信息可以让系统对同一用户的进程进行统一操作。

2. 处理器状态信息

这里指进程的运行现场保存区，进程在运行时，有很多现场信息存在于处理器的各种寄存器中，当进程在中断/异常进入内核程序运行及在内核程序中进一步调用子程序时，需要保存处理器运行现场，运行现场以栈帧格式保存在进程的核心栈中。所以从逻辑上说，进程核心栈属于进程控制块，当然在具体实现时并不一定将进程核心栈放在进程控制块当中，但进程控制块一定要有核心栈的地址。运行现场主要包括如下内容。

(1) 用户可用的寄存器或通用寄存器。这是指任意程序可以使用的数据或地址寄存器。

(2) 控制和状态寄存器。有许多用于控制处理器执行的寄存器，如包含下一执行指令地址的程序计数器(PC)、条件码寄存器(条件码是指当前逻辑或数学运算后导致进位或符号变化、溢出、全 0 或相等情况发生的标志，条件码寄存器即反映这种变化的寄存器)，还有包含中断是否开放、处理器执行态等状态信息的寄存器，通常称为程序状态字(PS 或 PSW，又可称“处理机状态字”)。

处理器状态信息包含所有处理器寄存器的内容，当进程正常运行时，这些信息存放于寄存器中；当进程在正常流程被外中断打断或异常发生时，寄存器信息必须被保护入栈，以便进程中断返回时可以恢复原来被中断的现场。保存处理器状态信息所涉及的寄存器取决于处理器硬件的设计，通常包含用户可见寄存器、控制和状态寄存器、栈指针等。特别要提到的是程序状态字(PS)，它记载了程序执行的状态信息，如条件码、外中断屏蔽标志、执行态(核心态还是用户态)标识等。

3. 进程控制信息

PCB 中还包括如下信息。

(1) 调度和状态信息。这些信息用于操作系统调度进程占用处理器，主要包括下列三项。

① 进程状态。定义进程当前的执行状况，如进程正在运行、就绪或等待状态。

② 调度相关信息。与操作系统调度算法相关，如优先级、时间片、本进程已等待时间及进程上次占用处理器时间等。进程调度程序可根据优先级决定进程占用处理器的优先次序，可以让等待时间长的进程优先占用处理器。

③ 等待事件。当进程处于等待状态时，指明进程所等待的事件，这往往是进程与系统或其他进程同步所引起的，如等待 I/O 结束、等待资源锁的释放、等待其他进程结束等。

(2) 进程间通信信息。为支持进程间通信相关的各种标识、信号、信件等，多个信件可以组织成队列。这些信息通常存放在信息接收方的进程控制块中。

(3) 存储管理信息，如进程映像的主存地址，在页式虚存系统中即是包含指向本进程页表结构的指针。

(4) 进程所用资源列表。说明由进程打开、使用的系统资源。如有指针指向打开的文件、I/O 设备等的描述、状态信息。

(5) 链信息。进程可以链接到一个进程队列中，或者链接到相关的其他进程中。例如，同一优先级的就绪进程被链成一个队列，一个进程可以链接它的父子进程，进程控制块需要有这样的一些指针域满足操作系统在处理同类、同簇进程时，对同类、同簇进程控制块的访问要求。

在操作系统中，每个进程都有一个唯一的内部数字标识符，它可以是进程控制块的地址值，或者是可以映射出进程控制块位置的某种系统内码。标识符是非常有用的。操作系统可以用进程标识符来定位进程控制块；当进程相互通信时，通过进程标识符可以说明要交换信息的对方进程；当进程创建子进程时，用进程标识符来指明父进程或子进程。这里的进程标识符是一个数字式的系统内码，通过它可以很快得出进程控制块的地址。

除进程标识符外，进程控制块中也包含相应的用户标识，指明进程所有者的信息或进程所运行程序的所有者的信息。同一用户的所有进程配合完成用户的上机意图。

进程控制块中的链接指针可以把有相同特性的进程控制块链接起来，如图 3.3 所示，有三个进程队列。因为只有一个处理器，所以运行队列只有一个进程，就绪队列存放等待处理器运行的进程，等待队列存放等待某种事件的进程，当然可以按照等待事件不同，设

立不同的等待队列，这样就可以用队列区分不同的事件了，待到事件发生，处理程序就到对应的等待队列中把进程的等待状态改成就绪状态。

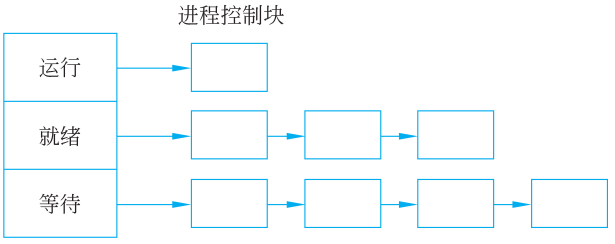

图 3.3　进程链结构

进程控制块是操作系统中最重要的数据结构，每个进程控制块都包含操作系统所需的进程信息。进程控制块几乎会被操作系统的所有模块访问和修改，包括各系统调用处理程序、进程调度、资源分配、中断处理和性能监督模块。

操作系统要直接访问这些表格并不难，每个进程都有唯一的标识符，它可以被用作输入，通过查找或变换得到 PCB 地址来访问进程控制块。但为了系统的层次化、模块化，可以利用面向对象设计方法，设计一个专门处理例程来处理对进程控制块的访问，所有其他程序都通过这个专门处理例程来读/写进程 PCB。可以在该处理例程中设置许多保护措施，而且在进程控制块结构调整时只需修改该例程即可。事实上，操作系统设计过程中，对所有公共数据结构都存在这样一个问题，用户往往把对该公共数据结构的规范操作集中在一个例程中，这称为**管程设计方法**。

3.2　进 程 状 态

进程表示程序的执行环境及执行过程。要做到程序的并发执行，操作系统必须能使不同进程中的程序占用处理器并运行，所以一个进程在从创建到结束的生命期内会占用处理器，也会从处理器上下来让别的进程去运行。

从处理器的观点来看，可以通过改变程序计数器(PC)的值来改变指令的执行次序。程序计数器可以从一个应用程序的代码段改变为另一个应用程序的代码段，这也是处理器进程切换所要做的事情。从各独立程序的角度看，各独立程序的执行被称为任务或进程运行。

可以把一个进程里的程序指令执行序列称为进程的踪迹。从处理器的角度来看，处理器的指令执行系列，是交替包含各进程踪迹的。假设有三个进程在一个处理器上运行，其处理器指令执行序列如图 3.4 所示。

首先，进程 1 的程序在处理器上执行，当因时钟中断进入内核，发现操作系统分给该进程的时间片用完时，进程调度程序会选取进程 2 来占用处理器；进程 2 执行到某系统调用，系统调用处理要向外设发送 I/O 请求，因其要等 I/O 完成，故进程调度程序会选取进程 3 来运行；当系统分给进程 3 的时间片用完后，进程调度程序又恢复进程 1 的上次保存现场往下执行。

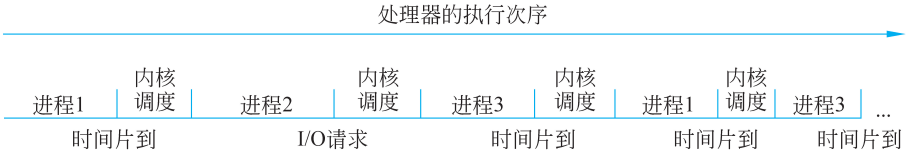

图 3.4 处理器指令执行序列

操作系统运行进程调度程序的时机是，当进程中断/异常从用户态进入核心态后，等待中断/异常处理完成，并准备返回用户态之前。例如，时钟中断发生并进入内核，用于处理与时间相关的事务，如时间片用完检测、系统时钟、处理定时器超时等。

从图 3.4 可看出，进程从创建到结束会经历运行、等待 I/O 完成、等处理器不同的状态，分别称为进程的运行、等待、就绪状态。

3.2.1 进程的创建与结束

进程是运行用户程序的实体。一般的进程都要经历创建、断断续续运行、最后结束的过程。当然，进程是不是结束要看如何利用进程来处理数据，有些情况下用进程来循环处理请求队列中的请求，那么这些进程循环地运行请求处理程序，永不结束，直到系统关机。这样的进程称为服务器进程或守护进程。

1. 进程创建

操作系统提供了进程创建的系统调用。用户程序可以通过“进程创建”系统调用创建新的进程去运行新的程序。创建新进程时，操作系统创建管理该进程所要的系统表格，为进程分配空间并初始化进程空间，准备好执行程序和数据。

一个进程通过系统调用创建一个新的进程，被创建的进程称为子进程，创建者进程称为父进程。除系统“祖先”进程外，其他进程只能由父进程建立。而“祖先”进程是在系统初始化时由初始化程序通过初始化“祖先”进程的进程控制块建立的。

在 UNIX/Linux 中，操作系统初始化时所创建的 1 号进程是所有用户进程的祖先，1 号进程为每个从终端登录系统的用户创建一个终端进程，这些终端进程又会利用“进程创建”系统调用创建子进程，从而形成进程层次体系，称为进程树或进程族系。

系统服务进程是一类特殊的进程，它们运行用户态程序，但不属于任何登录用户生成的进程，它们往往用于完成系统功能，如安全检查、网络服务等。UNIX/Linux 的 1 号进程也是一个系统服务进程，一旦创建，一般不会结束，除非系统要撤销它们提供的系统功能或要关机。

“进程创建”系统调用的一般处理过程如下。

(1) 接收新建进程运行所需的初始值、初始优先级、初始执行程序描述等由父进程传来的参数。

(2) 请求为进程控制块(PCB)分配空间，得到一个内部数字进程标识。

(3) 用从父进程传来的参数初始化 PCB 表。

(4) 产生描述进程空间的数据结构，如页表；用初始参数指定的执行文件初始化进程

空间，建立程序区、数据区、栈区等。

（5）用进程运行初始值，初始化该进程的处理器现场保护区。构造一个现场栈帧。等该进程第一次被调度后会从该栈帧恢复现场，从而能够进入用户程序的入口点运行。

（6）设置好父进程等关系域。

（7）将进程置成就绪状态。

（8）将 PCB 表挂入就绪队列等待被调度运行。

2. 进程结束

操作系统为用户提供系统调用服务用于结束进程，以释放进程所占用的所有系统资源。进程可以请求操作系统结束自己，或在进程发生异常时陷入内核并结束进程。进程结束处理主要是释放进程所占用的系统资源，进行有关信息的统计工作，理顺本进程结束后其他相关进程的关系，最后调用进程调度程序选取高优先级就绪进程来运行。

“进程结束”系统调用的处理过程如下。

（1）将进程状态改到结束状态。

（2）关闭所有打开、使用的文件、设备。

（3）脱离用户进程与其所执行程序文件的映射关系。

（4）统计相关信息，将统计信息记入 Log 文件或进程控制块中。

（5）清理其相关进程的链接关系，如在 UNIX/Linux 中，将该结束进程的所有子进程链接到 1 号进程上，作为 1 号进程的子进程，并通知父进程自己结束。

（6）释放进程空间，释放进程控制块空间。

（7）调用进程调度程序将处理器转移到其他进程运行。

3.2.2　进程状态变化模型

进程在它的生存周期中，由于系统中各进程并发运行及对资源独占访问等相互制约，使得它们的状态不断发生变化。通常进程处于以下 5 种状态。各操作系统具体实现不同，可以进一步细分就绪和等待状态。

（1）运行状态（Running）：一个进程正在处理器上运行，称此进程处于运行状态。在单机环境下，每一时刻最多只有一个进程处于运行状态。

（2）就绪状态（Ready）：一个进程获得了除处理器之外的一切所需资源，一旦得到处理器即可运行，称此进程处于就绪状态。

（3）等待状态，又称阻塞状态（Blocked）：一个进程正在等待某个事件而暂停运行，如等待某个资源成为可用或等待 I/O 完成。

（4）创建状态（New）：一个进程正在被创建，是还没转到就绪状态之前的状态。

（5）结束状态（Exit）：一个进程正在从系统中消失时的状态。这是因为进程结束或其他原因所致。

创建状态和结束状态对系统进程管理是非常有用的。当一个新进程被创建时，为它分配了进程控制块结构，置上了进程标识等参数，系统已经存在该进程的标识表格了，但它还不能调度运行，这时需要置上该进程为创建状态，表示该进程的表格还需要完成进一

步的初始化工作才可以调度运行。从某种意义上说，置上创建状态保证了进程创建对进程控制结构操作的完整性。等到创建工作完成，再将进程状态置为就绪状态，这时进程调度程序才可以选取该进程，以保证进程调度一定是在创建工作完成后进行。

同样地，当进程到达自然结束点结束，或因某种错误和由其他方式授权进程结束时，进程不能再运行，系统需要逐步释放系统资源，最后释放进程控制块。因此，系统首先必须置该进程处于结束状态，再进一步做信息统计、资源释放工作，最后将进程控制块表格清零，并将表格存储空间返还系统。

图 3.5 说明了进程状态变化的情形，可能的状态变化如下。

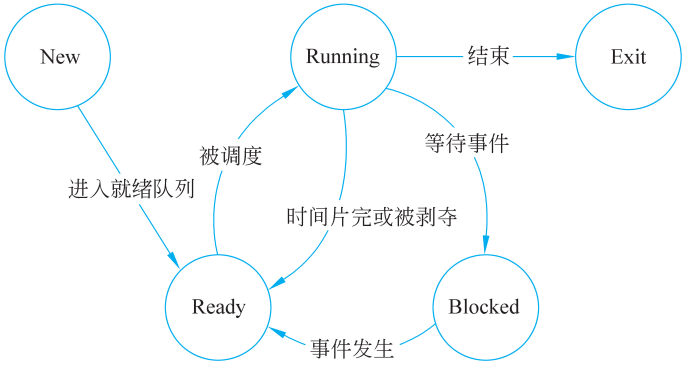

图 3.5　进程状态变化图

(1) 空→创建状态。当产生一个新进程用于执行一个程序时，新进程处于创建状态。产生进程最先发生在操作系统初始化时，由系统初始化程序为系统创建第一个进程，以后由父进程通过创建进程的系统调用产生子进程。

(2) 创建状态→就绪状态。当进程创建完成并初始化后，一切就绪，准备运行时变为就绪状态。

(3) 就绪状态→运行状态。处于就绪状态的进程被进程调度程序选中，然后被分配到处理器上来运行，于是进程状态变成运行状态。

(4) 运行状态→结束状态。当进程发出结束进程系统调用或者因错需提前结束时，当前运行进程会进入内核做结束处理。

(5) 运行状态→就绪状态。处于运行状态的进程在其运行过程中，时钟中断处理程序发现分给它的时间片用完，从运行状态变为就绪状态。进程还有以下可能出现这种状态的改变：在可剥夺的操作系统中，当某中断处理程序使更高优先级的进程就绪后，调度程序可以将正运行进程从运行状态改变为就绪状态，选更高优先级的进程运行。

(6) 运行状态→等待状态。当进程请求某个资源且必须等待得到它时，就从运行状态变为等待状态，进程往往以系统调用的形式请求操作系统提供服务，这是一种特殊的、由运行的用户态程序调用内核程序的形式。例如，当进程请求操作系统服务时，操作系统得不到提供服务所需的资源；或进程请求一个 I/O 操作时，操作系统已启动外部设备，但 I/O 操作尚未完成；或进程与其他进程通信，要接收信件时，但对方还未发出，进程都会阻塞而进入等待状态。

（7）等待状态→就绪状态。当进程要等待的事件到来时，如发生 I/O 结束中断，中断处理程序必须把相应的 I/O 结束的进程状态从等待状态变为就绪状态。

现代操作系统不再像早期的操作系统一样依次排列进程控制块，由进程状态标志来区分不同的状态，查找时还须顺序扫描。现代操作系统依靠队列把相关的进程链接起来，以节省系统查找进程的时间。图 3.6 说明了进程创建后状态变化时，操作系统管理的队列结构及变化图。

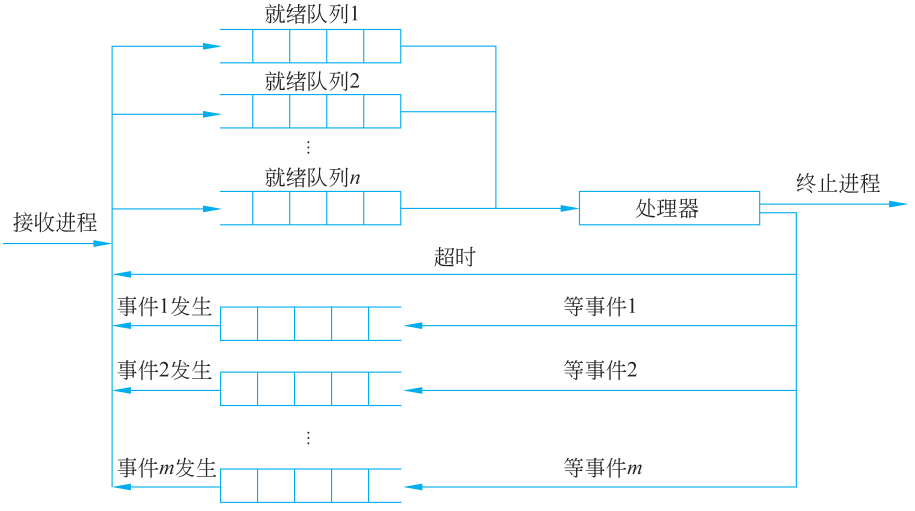

图 3.6　进程创建后状态变化时，操作系统管理的队列结构及变化图

系统按进程优先级数设立几个就绪进程队列，同一优先级进程在同一队列中。系统先取最高优先级的队列队首进程占用处理器，当时间片用完时，往往会重新计算优先级并将其挂到相应就绪队列中。等待事件时，将其挂到相应的事件等待队列中。如果某个事件发生，则系统从相应等待队列中选取进程，重新计算优先级并将其挂到就绪队列中。

3.3　进程控制与调度

3.1 节和 3.2 节描述了进程在系统中的表示及进程的状态变化。下面介绍进程运行和切换的实现技术。

3.3.1　进程执行

在创建进程时，操作系统为它的运行建立了栈帧，准备好了初始现场，一旦被进程调度程序选择占用处理器运行，调度程序会马上把栈中存放的初始现场信息恢复到处理器的各个寄存器中，在存放输入参数的寄存器中放置由栈中得到的初始输入参数，进程运行程序的初始地址也恢复到 PC 中。进程会运行创建进程时指定的运行程序。创建进程时指定运行的程序是由进程在用户态下运行的，如果进程在运行程序的过程中发生了中断或异常（如系统调用、外部设备中断），进程会转入内核程序运行。

1. 执行模式

处理器在执行用户程序和执行内核程序时的模式是有区别的，这是因为处理器在运行内核程序时，可以获得更多的特权，以便内核程序实现更强的功能。而在运行用户程序的时候，其权限有限，也不能直接访问操作系统内核空间，这样保障了系统的安全性。许多处理器支持至少两种执行态。某些指令只能在特权态下执行，如读/写程序状态字(PS)等控制寄存器及存储管理相关的一些指令。另外在特权态下，程序还可以访问更大的地址空间。

用户态是非特权模式，用户程序在这一模式下运行。当然，现代操作系统的许多系统功能也由运行在用户态的程序实现，如 UNIX/Linux 操作系统的 1 号进程，它在用户态运行 INIT 程序，负责所有用户终端进程的创建。特权态又称核心态、系统态或监督模式。内核程序在这种模式下运行。

以往在特权状态才能执行的一些指令在现代计算机中变成了通用指令。它们在任何模式下都可以执行，如 I/O 指令。现代计算机可用通用的读/写指令来实现 I/O 操作，它与一般读/写指令不同的是，它的物理地址指向一片特定的空间，当指令译码发现是对特定空间的访问时，把它转送 I/O 控制部件，做相应的 I/O 操作。如果操作系统提供某种手段能将指定物理空间映射到用户(虚存)空间，那么不但能实现用户态的 I/O 驱动程序，而且还能用高级语言实现 I/O 操作，这是因为 I/O 操作与一般读/写指令格式一致，所不同的只是地址范围不一样。

划分特权与非特权态的理由是，保护操作系统和操作系统的数据表格不被可能出错的用户程序破坏。处理器如何知道它在哪种态下？执行态如何变化？何时变化呢？前面讲到过，程序状态字(PS)中有一位表示处理器执行态，当运行在处理器上的用户程序产生自陷，或被外部事件中断进入操作系统程序运行时，系统要通过重置 PS 马上将处理器模式转变为核心态，原程序状态字作为运行现场被保护，新的程序状态字中的状态被置为核心态。这时，处理器执行任何特权指令或访问系统的空间都不会报错，待到返回用户态程序时，原保护的程序状态字被恢复，处理器模式又转到用户态下。此后如果执行了特权指令，处理器则会报错。

如上所述，进程既为运行用户程序而创建，又会在用户自陷或外部中断时去运行内核程序。当进程运行内核程序时，系统只是保存了用户程序的运行现场，包括所有处理器状态、现场信息，保留原用户程序使用的用户栈不被核心态程序使用。当进程运行内核程序时，系统使用为该进程分配的核心栈空间，当内核程序调用子程序或被中断时，可以利用核心栈保存现场。

2. 态切换

处理器在执行每条机器指令时，都会查看是否有中断发生。如果没有中断，处理器继续原来的执行流程；如果有中断发生，处理器会按设计时的约定，用中断处理程序的入口地址重置 PC，并重新设置 PS 将处理器模式从用户态置成核心态模式，转向操作系统的程序执行。操作系统应马上继续保护中断点的处理器现场。在处理器控制权转到内核入

口后，内核入口程序应做以下工作。

（1）保护处理器现场，硬件已保护原程序计数器、程序状态字，软件继续保存各种其他寄存器，如用户栈的指针。

（2）处理器模式此时已转换成核心态，以便以后执行的程序可以运行特权指令。

（3）根据中断级别设置中断屏蔽。一般地，如果发生了某级中断，则要屏蔽该级及以下级别的中断，转中断处理程序。

用户程序执行 trap 指令进入内核运行的情形与上述情形基本相似，只是无须设置中断屏蔽，然后操作系统调用服务例程运行。

在操作系统处理中断或系统调用过程中，可能会导致一些等待状态的高优先级进程变成就绪态，如一个 I/O 结束中断处理后会使原来那个等 I/O 完成的等待进程变成就绪。在处理时钟中断的过程中，可能会发现正运行的进程时间片已用完，这都会请求进程的调度及切换。当上述情形发生时，操作系统立即置好“请求进程调度”标志，马上进行进程调度并切换，或等到中断或异常处理完成准备返回恢复点时进行进程调度并切换。当然，如果没有发生上述事情，原进程的现场还会被恢复运行。

3. 进程切换

进程切换是指处理器从一个进程的运行转到另一个进程上运行。进程切换与处理器模式切换不同，当模式切换时，处理器逻辑上还在同一进程中运行。而进程切换指处理器转入另一个进程运行。

如果进程因外部中断或异常进入核心态运行，在执行完操作后又恢复用户态刚被中断的程序运行，则操作系统只需恢复进程进入内核时所保存的处理器现场，无须改变当前进程空间等环境信息。但如果要切换进程，那么当前运行进程改变了，当前进程空间等环境信息也需要改变，如当前进程页表始地址寄存器就需要改成新进程的页表始地址。从一个进程切换到另一个进程的过程大致如下。

（1）保存处理器的上下文，包括程序计数器（PC）、程序状态字（PS）、进程核心栈指针等其他寄存器。

（2）修改当前运行进程的进程控制块内容，包括将进程状态从运行态改成其他状态，将该进程的进程控制块链接到相应新状态队列中。

（3）确定被调度进程。

（4）修改被调度进程的进程控制块，包括把其状态改变为运行态。

（5）将当前进程存储管理数据修改为新选进程的存储管理数据。例如，改变当前进程存储地址寄存器的内容。如果是虚存系统，由于原来在处理器快表（TLB）中的页表项都是原进程页表项的缓冲，其地址变换是原进程的虚实映射，所以要作废掉处理器快表中的各项数据，并将当前进程页表指针改为指向新选定的进程页表。

（6）恢复被选进程上次切换出处理器时的处理器现场，按原保存的程序计数器值重置程序计数器，运行新选进程。

如图 3.7 所示为模式切换与进程切换时机。

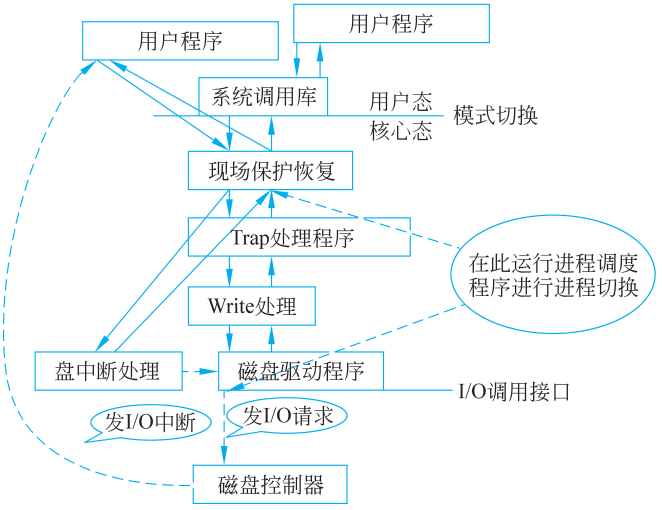

图 3.7 模式切换与进程切换时机

3.3.2 进程调度

在单处理器多道程序设计系统中，进程被作为占用处理器运行的执行单位。由于处理器是最重要的计算机资源，所以合理、有效地选择进程占用处理器是进程调度的重要任务。提高处理器的利用率及改善系统响应时间、吞吐率（单位时间完成的作业量），在很大程度上取决于进程调度性能的好坏。

选择进程占用处理器的调度又称为低调或低级调度。一个进程能够在处理器上运行之前，还必须占有系统的其他资源（如主存等）。为了合理地安排进程占用这些资源，为进程占用处理器运行做准备，操作系统也存在对其他资源调度的概念，即选择进程占用系统的其他资源。

1. 一般调度的概念

一般意义上，调度就是选择。操作系统管理系统的有限资源，当有多个进程（或多个进程发出的请求）要使用这些资源时，鉴于资源的有限性，必须按照一定的原则选择进程（或请求）来占用资源，这就是调度。选择进程占用处理器称为进程调度；选择磁盘请求占用磁盘控制器进行磁盘 I/O 操作称为磁盘调度。有时因为使用者太多，被使用的资源太少，会设置一种“使用资格”状态，当选择资源申请者进入“使用资格”状态时也称为调度。

下面介绍 3 种针对使用者占用不同资源的调度。

（1）高级调度，又称作业调度。在支持批处理的系统中，新提交的作业排入批处理队列中。高级调度从批处理队列中选取合适的作业，产生相应进程运行作业控制语句的解释程序，作业一旦被高级调度选中，相应的进程及进程组才会产生，其他系统资源才有可能被占用。

（2）中级调度。中级调度选取进程占用物理主存，为占用处理器做好准备。在虚存

方式存储管理系统中，如果进程被中级调度选中，进程才有资格占用主存。对于在主存不够时被交换到辅存的进程，系统将进程占用的所有主存页帧释放，且在没被中级调度选中之前不会再占用主存。因此，通过中级调度可以控制进程对主存的使用。一个进程在生命周期内可能被多次交换。

(3) 低级调度。低级调度决定处在就绪状态中的哪个进程将获得处理器。通常所说的进程调度就是指低级调度。执行低级调度往往使原占用处理器的进程因各种原因放弃处理器，如有更高优先级的进程变成就绪状态的原因。实现低级调度的程序往往又被称为分派程序(Dispatcher)。

下面重点讲述进程调度问题，将讨论进程调度的剥夺与非剥夺方式、请求调度的因素、调度时机和进程切换时机。为了实现系统响应时间、吞吐率及资源利用率等方面的指标，设计进程调度时不但要考虑进程调度算法，而且要考虑提供一个好的调度机制。

2. 进程调度方式

进程调度有以下两种基本方式。

(1) **非剥夺方式**。在这种方式下，一旦分派程序把处理器分配给某进程后，便让它一直运行下去，直到进程完成或发生某事件(如提出 I/O 请求)而阻塞时，才把处理器转给另一进程。这种调度方式的优点是简单、系统开销小。但它存在很明显的缺点，如当一个紧急任务到达时，不能立即投入运行，实时性差。如果有若干后到的短进程，则需要等待长进程运行完毕，致使短进程的周转时间变长。

例如，有三个进程(P_1，P_2，P_3)先后(但又几乎在同时)到达就绪状态，它们分别需要 20，4 和 2 个单位时间运行完毕，若它们按 P_1，P_2，P_3 的顺序执行且不可剥夺，则三个进程各自的周转时间分别为 20，24 和 26 个单位时间，这种非剥夺方式对短进程 P_3 而言是不公平的。

(2) **剥夺方式**。在这种方式下，某个进程正在运行时因中断或系统调用进入内核后，可以被系统以某种原则剥夺它的处理器，将处理器转给其他进程。剥夺原则可以是优先权原则，即高优先级进程可以剥夺低优先级的进程运行；也可以是时间片轮转原则，在运行进程的时间片用完后被剥夺处理器。

图 3.8 给出了前述的三个进程在基于时间片轮转原则的剥夺调度方式下的运行情况(假设时间片值为两个时间单位)。可以看出，P_1，P_2，P_3 的周转时间分别为 26，10，6 个单位时间，平均周转时间已由非剥夺方式的 23.5 降低为 14 个单位时间。采用剥夺方式的调度，对加快系统吞吐率、加速系统响应时间都有明显的好处。

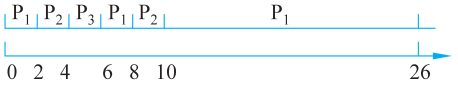

图 3.8　基于时间片轮转原则的剥夺调度方式

3. 引发进程调度的原因

当进程正常运行时，怎么会出现进程调度的要求呢？这主要有两方面的原因：一方面，正在运行的进程无法再继续运行下去，而系统中有其他待运行的进程，操作系统必须把处理器交给一个合适的进程来运行；另一方面，对可剥夺调度方式的支持，例如，出现因为更高优先级的进程从原来的等待状态变为就绪状态，需要处理器，或为了给等待处理器过长的进程占用处理器，从而引起调度。

进程主动放弃处理器的原因如下。

（1）正在执行的进程执行完毕。当运行程序的进程执行完程序要结束其使命时，它向操作系统发“进程结束”系统调用，操作系统在处理“进程结束”系统调用后应请求重新调度。

（2）正在执行的进程发出 I/O 有关的系统调用，当操作系统内核外设驱动程序启动外部设备 I/O 后，在 I/O 请求没有完成前要将进程变成等待状态，这时应该请求重新调度。

（3）正在执行的进程要等待其他进程或系统发生的事件，如 receive 系统调用等待另一个进程的通信数据，这时操作系统应将正运行进程挂到等待队列，并且请求重新调度。

（4）正在执行的进程得不到所要的系统资源，如要求进入临界段，但还不能进入时，这时等待的进程应放弃处理器，并且请求重新调度。

（5）正在执行的进程执行“自愿放弃处理器”的系统调用时。有些操作系统提供这种系统调用。

为了支持可剥夺的进程调度方式，在以下情况发生时，新就绪的进程可能会按某种调度原则替换正运行的进程，因此也应该申请进行进程调度。

（1）当中断处理程序处理完中断，如 I/O 中断、通信中断，导致等待该数据的进程变成就绪状态时，应该请求重新调度。

（2）当进程释放资源，走出临界段，导致其他等待该资源进程从等待状态进入就绪状态时，应该请求重新调度。

（3）当进程发出系统调用，引起某个事件发生，导致其他等待事件的进程就绪时。如 send 系统调用给某进程发送了消息。

（4）其他任何原因导致有进程从其他状态变成就绪状态，如进程被中级调度选中时。

为了支持可剥夺调度，即使没有新就绪进程，为了让所有就绪进程轮流占用处理器，可在下述情况下申请进行进程调度。

（1）当时钟中断发生，时钟中断处理程序调用有关时间片的处理程序，发现正在运行的进程的时间片用完时，应请求重新调度，以便让其他进程占用处理器。

（2）在按进程优先级进行进程调度的操作系统中，任何原因导致进程的优先级发生变化时，应请求重新调度，如进程通过系统调用自愿改变优先级，或者系统处理时钟中断时，或者根据各进程等待处理器时长而调整进程的优先级时。

操作系统并不一定在引发进程调度原因产生时就马上运行进程调度程序。下面说明进程调度与切换的时机。

4. 进程调度和切换时机

进程调度和切换程序是属于内核的程序。当请求调度的事件发生后，才可能运行进程调度程序。当选中了新的就绪进程后，才会进行进程间的切换。这三件事情必须按这个顺序执行，但是它们并不一定是连续执行的。从理论上讲，这三件事情应该一气呵成，但在实际设计中，如果在某个时刻发生了引起进程调度的因素，并不一定能够马上进行调度与切换。

例如，在运行中断处理程序时，来了一个更高优先级的 I/O 中断，在高优先级中断处理完成后，因为把对应的等待该 I/O 中断的进程变为就绪状态，所以请求调度。如果此时马上进行进程调度与切换，则原被高级中断打断的那个低级中断处理程序还没有运行完成，不但影响中断的响应时间，原来在 I/O 硬件中的现场也可能会丢失。再如，当进程在临界段中时，发生时钟中断，导致请求调度事件发生。如果此时马上进行进程调度与切换，则会导致临界段所占资源不能尽快被释放。

在现代操作系统设计中，不能进行进程调度和切换的情况如下。

(1) 在处理中断的过程中。此时，部分现场存在于外部设备控制器中，当时的外部设备状态也是现场的一部分，如果这时进行进程切换，必须保存当时的一切现场，这在实现上很难做到，而且中断处理是系统工作的一部分，逻辑上不属于某个进程，不应作为某进程的程序段而被剥夺处理器。

(2) 其他需要完全屏蔽中断的原子操作过程中。在原子操作过程中，连中断都要屏蔽，更不应该进行进程调度与切换。关于原子操作详见 4.2.3 节中的原语概念与实现。

(3) 当进程在临界段中时，一般也不应该被切换。在运行用户程序的进程 trap 进入操作系统后，如果进入临界段，需要独占式访问共享数据，理论上必须加锁，以防其他并行程序进入。在加锁后到解锁前不应切换到其他进程运行，以加快该共享数据的释放。关于临界段详见第 4 章。

如果在上述过程中发生了引起调度的条件，比如就绪了某个进程，并不能马上进行调度和切换，系统只好置上请求调度标志，等到走出上述过程后再来调用调度程序进行相应的调度与切换。操作系统何时进行进程调度和切换与操作系统的实现相关。

应该进行进程调度与切换的时机如下。

(1) 当发生引起调度条件，且当前进程无法继续运行下去时(如发生各种进程放弃执行的条件)，可以马上进行调度和切换。如果操作系统只在这种情况下进行进程调度，那么该操作系统支持非剥夺调度。

(2) 当中断处理结束或 trap 处理结束后，返回被中断进程的用户态程序执行现场前，若置上请求调度标志，即可马上进行进程调度和切换。如果操作系统支持这种情况下运行调度程序，即实现了剥夺方式的调度。因为这时原进程并没有主动放弃处理器，而是在准备返回用户态程序继续执行时悄悄运行了调度程序而被剥夺的。

如图 3.7 所示的调度时机一个发生在返回用户态执行前，一个发生在向外设控制器发出 I/O 请求进程阻塞后。切换往往在调度完成后立刻发生。进程的切换过程根据处理器结构的不同而不同。典型的进程切换要求保存原进程当前切换点的现场信息，恢复

被调度进程的现场信息，现场信息主要有 PC、PS、其他寄存器内容、核心栈指针、进程存储空间的指针（如进程页表指针寄存器）。

为了进行进程现场切换，内核将原进程的上述信息推入当前进程的核心栈来保存它们，并更新堆栈指针。内核从新进程的核心栈中装入新进程的现场信息，还要更新当前运行进程存储空间指针，并且作废掉处理器快表中有关原进程的信息。在内核完成清除工作后，重新设置 PC 寄存器，控制转到新进程的程序，新进程开始运行。

3.3.3 调度算法

前面已经描述了进程的调度时机，本节将介绍常用的进程调度原则和算法。在描述进程调度算法之前，先给出几个概念。

（1）周转时间。进程从创建到结束运行所经历的时间。

（2）平均周转时间。n 个进程的周转时间的平均值。一般来说，如果调度算法使得平均周转时间减少，则用户满意度和系统效率会提高。

（3）等待时间。指进程处于等待处理器状态的时间之和。等待时间越长，用户满意度越低。

（4）平均等待时间。n 个进程的等待时间的平均值。如果一个调度算法使得平均等待时间短了，意味着减少了平均周转时间。

算法的时间和空间开销也是要考虑的一个主要因素。不同的调度算法可满足不同的要求，要想得到一个满足所有用户要求且开销小的算法，几乎是不可能的。

1. 先来先服务调度算法

先来先服务调度算法（FCFS）是最简单的调度方法。其基本原则是，按照进程进入就绪队列的先后次序进行选择。对于进程调度来说，一旦一个进程占有了处理器，它就一直运行下去，直到该进程完成其工作或者因等待某事件而不能继续运行时才释放出处理器。先来先服务调度算法属于不可剥夺算法。

从表面上看，这个方法对于所有进程都是公平的，并且一个进程的等待时间是可以预先估计的。但是从另一方面来说，这个方法并非公平，因为当一个大进程先到达就绪状态时，就会使许多小进程等待很长时间，增加了进程的平均周转时间，会引起许多小进程用户的不满。

今天，先来先服务调度算法已很少用作主要的调度，尤其是不能在分时和实时系统中用作主要的调度算法。但它常被结合在其他的调度中使用。例如，在使用优先级作为调度的系统中，往往对许多具有相同优先级的进程使用先来先服务的原则。

2. 短进程优先调度算法

短进程优先调度算法（SPF）从进程的就绪队列中挑选那些所需运行时间（估计时间）最短的进程进入主存运行。这是一个非剥夺式算法，它一旦选中某个短进程，就将保证该进程尽可能快地完成运行并退出系统。这样就减少了在就绪队列中等待的进程数，同时也降低了进程的平均等待时间，提高了系统的吞吐量。但从另一方面来说，各个进程的等

待运行时间的变化范围较大，并且进程（尤其是大进程）的等待运行时间难以预先估计。也就是说，用户对他的进程什么时候完成心里没有底，当后续短进程过多时，大进程可能没有机会运行，导致饿死。而在先来先服务算法中，进程的等待和完成时间是可以预期的。

短进程优先调度算法要求事先能正确地了解进程将运行多长时间。但通常一个进程没有这方面可供使用的信息，只能估计。

短进程优先算法和先来先服务算法都是非剥夺的，因此均不适合于分时系统，因为不能保证对用户及时响应。

3. 最短剩余时间优先调度算法

最短剩余时间优先调度算法是将短进程优先调度算法用于分时环境的变形。其基本思想是，让“进程运行到完成时所需的运行时间最短”的进程优先得到处理，包括新进入系统的进程。在短进程优先调度算法中，一个进程一旦得到处理器就一直运行到完成（或等待事件）而不能被剥夺（除非主动让出处理器）。而最短剩余时间优先调度算法允许被一个新进入系统且其运行时间少于当前运行进程的剩余运行时间的进程所抢占。

该算法的优点是，可以用于分时系统，保证及时响应用户要求。缺点是，系统开销增加。首先，要保存进程的运行情况记录，以比较其剩余时间大小；其次，剥夺本身也要消耗处理器时间。毫无疑问，这个算法使短进程一进入系统就能立即得到服务，从而降低进程的平均等待时间。

4. 最高响应比优先调度算法

Hansen 针对短进程优先调度算法的缺点提出了最高响应比优先调度算法。这是一个非剥夺的调度算法。按照此算法，每个进程都有一个优先数，该优先数不但是要求的服务时间的函数，而且是该进程得到服务所花费的等待时间的函数。进程的动态优先数计算公式如下。

$$\text{优先数}=(\text{等待时间}+\text{要求的服务时间})/\text{要求的服务时间}$$

要求的服务时间是分母，所以对短进程是有利的，它的优先数高，可优先运行。但是由于等待时间是分子，所以长进程由于其等待了较长时间，从而提高了其调度优先数，终于被分给了处理器。进程一旦得到了处理器，它就一直运行到进程完成（或因等待事件而主动让出处理器），中间不被抢占。

可以看出，“等待时间＋要求的服务时间”是系统对作业的响应时间，所以优先数公式中，优先数值实际上也是响应时间与服务时间的比值，称为响应比。响应比高者得到优先调度。

5. 优先级调度算法

按照进程的优先级大小来调度，使高优先级进程优先得到处理器的调度称为优先级调度算法。进程的优先级可以由操作系统按一定原则赋予它，如实时进程给更高的优先级，甚至可由用户支付高额费用来购买优先级。

但在许多采用优先级调度的系统中，一般进程采用动态优先级。一个进程的优先级不是固定的，可能会随许多因素的变化而变化，例如，进程的等待时间长优先级变高、已使用的处理器时间长了优先级变低。其实这种可变优先级算法是一种反馈算法。

优先级调度算法又可分为下述两种。

(1) 非剥夺的优先级调度算法。一旦某个高优先级的进程占有了处理器，就一直运行下去，直到由于其自身的原因而主动让出处理器时(任务完成或等待事件)才让另一高优先级进程运行。

(2) 可剥夺的优先级调度算法。任何时刻都严格按照高优先级进程在处理器上运行的原则进行进程的调度，或者说，在处理器上运行的进程永远是就绪进程队列中优先级最高的进程。在进程运行过程中，一旦有另一个优先级更高的进程出现(如一个高优先级的等待状态进程因事件的到来而成为就绪)，进程调度程序就迫使原运行进程让出处理器给高优先级进程使用或称为抢占了处理器。在现代操作系统中，其一般进程调度算法就属于“可剥夺的优先级调度算法”，每个进程的优先数都是动态优先数，由系统为各进程每隔一个时间间隔计算一次优先数。

6. 时间片轮转算法

采用此算法的系统其进程就绪队列往往按进程到达的时间来排序。进程调度程序总是选择就绪队列中的第一个进程，也就是说，按照先来先服务原则调度，但进程仅占有处理器一个时间片，在使用完一个时间片后，进程还没有完成其运行，它也必须释放(被剥夺)处理器给下一个就绪的进程。而被剥夺的进程返回到就绪队列的末尾重新排队，等候再次运行。时间片轮转算法特别适合于分时系统使用，当多个进程驻留在主存时，在进程间转接的开销一般不是很大。

由于时间片值对计算机系统的有效操作影响很大，所以在设计此算法时，应考虑下列问题：时间片值如何选择？它是固定值还是可变值？它对所有用户都相同还是随不同用户而不同？显然，如果时间片值很大，大到一个进程足以完成其全部运行工作所需的时间，那么此时间片轮转算法就退化为先来先服务算法了。如果时间片值很小，那么处理器在进程间的切换工作过于频繁，使处理器的切换开销变得很大，而处理器真正用于运行用户程序的时间将会减少。通常，最佳的时间片值应能使分时用户得到好的响应时间，因此时间片值应大于大多数分时用户的询问时间，即当一个交互进程正在执行时，给它的时间片值相对来说略大些，使它足以产生一个 I/O 请求；或者时间片值略大于大多数进程从计算到 I/O 请求之间的间隔时间。这样可使用户进程工作在最高速度上，并且也减少了进程间切换的不必要的开销，提高了处理器和 I/O 设备的利用率，同时也能提供较好的响应时间。

各个系统的最佳时间片值是不同的，而且随着系统负荷不同而有所变化。关于时间片值的更进一步考虑和时间片轮转算法请参阅“多级反馈队列调度算法”。

特别要注意的是，时间片是否用完的判定程序是由时钟中断处理程序激活的，因此时间片值必须大于时钟中断间隔。

7. 多级反馈队列调度算法

这里所研究的算法是时间片轮转调度算法的发展，但是本算法同时考虑：

（1）为提高系统吞吐量和降低进程的平均等待时间而照顾短进程。

（2）为得到较好的 I/O 设备利用率和对交互用户的及时响应而照顾 I/O 型进程。

（3）在进程运行过程中，按进程运行情况动态地考虑进程的性质（I/O 型进程还是处理器型进程）。并且要尽可能快地决定进程当时的运行性质（以 I/O 为主还是以计算为主），同时进行相应的调度。

具体来说，多级反馈队列的概念如图 3.9 所示。系统中有多个进程就绪队列，每个就绪队列对应一个调度级别。第 1 级队列的优先级最高，以下各级队列的优先级逐次降低。调度时选择高优先级的第 1 个就绪进程。

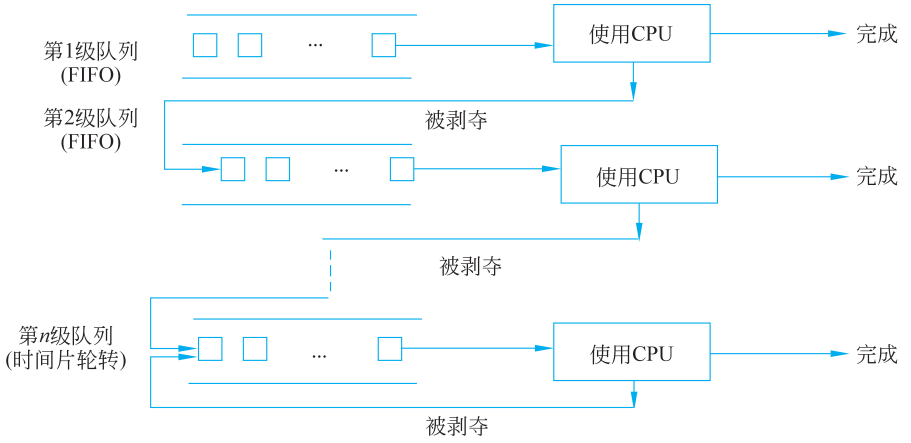

图 3.9　多级反馈队列的概念

各级就绪队列中的进程具有不同的时间片值。优先级最高的第 1 级队列中的进程的时间片值最小，随着队列级别的增加，其进程的优先级降低了，但时间片值却增加了。通常下放一级，其时间片值增加 1 倍。各级队列均按先来先服务原则排序。

调度方法：当一个新进程进入系统后，它被放入第 1 级就绪队列的末尾。该队列中的进程按先来先服务原则分给处理器，并运行一个时间片。假如进程在这个时间片中完成了其全部工作，或因等待事件或等待 I/O 而主动放弃了处理器，该进程撤离系统（任务完成）或进入相应的等待队列，从而离开就绪队列。若进程使用完了整个时间片后，其运行任务并未完成（也没有产生 I/O 请求），仍然要求运行，则该进程被剥夺处理器，同时它被放入下一级就绪队列的末尾。当第 1 级进程就绪队列为空后，调度程序才去调度第 2 级就绪队列中的进程。其调度方法同前。当第 1、2 级队列全部为空时，才调度第 3 级队列中的进程……当前面各级队列全部为空时，才调度最后第 n 级队列中的进程。第 n 级（最低级）队列中的进程采用时间片轮转算法进行调度。当比运行进程更高级别的队列到来一个新的进程时，它将抢占运行进程的处理器，而被抢占的进程回到原队列的末尾。

多级反馈队列的调度操作如上所述，它根据进程执行情况的反馈信息而对进程队列