第3章 进程和处理机管理

操作系统的最重要的特征之一是程序的并发执行，为了描述和管理程序的执行，提高系统效率和增强系统内各种硬件的并行执行，通常引入进程概念。在现代操作系统中，进程是设计和分析操作系统的重要工具，资源分配是以进程为单位，系统运行也是以进程为基本单位，在操作系统中进程这个概念极为重要，因此本章从进程的角度来研究操作系统，最后研究了 Windows XP 中的进程管理。

3.1 进程的基本概念

用静态的观点看操作系统是一组程序和表格的集合，用动态的观点看操作系统是进程的动态和并发执行。而进程的概念实际上是程序这一概念的发展产物。因此，我们从分析程序入手，引出“进程”的概念。

程序的执行可分为顺序执行和并发执行两种方式。顺序执行是指操作系统依次执行各个程序，在一个程序的整个执行过程中该程序执行占用所有系统资源，不会中途暂停。顺序执行是单道批处理系统的执行方式，也用于简单的单片机系统。并发执行是指多个程序在一个处理机上的交替执行，这种交替执行在宏观上表现为同时执行。现代操作系统多采用并发执行方式，因而具有许多新的特征。引入并发执行的目的是提高计算机资源利用率。

3.1.1 程序顺序执行

1. 程序的顺序执行

人们使用计算机的目的就是对数据进行加工处理，以此来解决各类问题。数据就是那些用来表示人们思维对象的抽象概念的物理表现，而经过解释和处理以满足特定需要的数据叫做“信息”。数据是在人与人之间、人与计算机之间传递信息的，可以存储起来供将来使用，也可以用来按某种规则予以处理以导出新的信息。

数据处理的规则叫做操作。每个操作都要有执行对象，一经启动将在一段有限的时间内执行完毕，并能根据状态的变化辨认出操作的结果。

对某一有限数据集合所实施的、目的在于解决某一问题的一组有限的操作的集合，称为一个计算。即计算是由若干个操作组成的。程序是算法的形式化描述，所以一个程序的执行过程就是一个计算。

一个程序通常是由若干个程序段组成的，一个程序段对应一个操作，这些操作必须有严格的先后顺序，仅当前一个操作执行完毕，才能执行后面的操作。

例如，一个程序在执行时，总是可以抽象为先输入用户的程序和数据，再计算，最后打印输出计算结果。如果用结点 I 表示输入操作、C 表示计算操作、P 表示输出操作，则一个程序的执行顺序就是 I、C、P。当一个程序执行完后，下一个程序才可以开始运行，如图 3-1 所示，其中，下标表示作业编号。

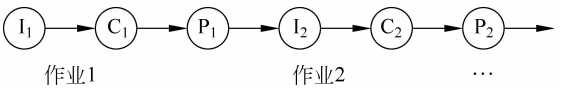

图 3-1　程序的顺序执行

另外的例子是程序源代码中顺序执行的语句，只有一个语句执行完后，后面的语句才可以开始执行。

2. 程序顺序执行的特点

程序顺序执行的操作是一个接一个以有限的速度向前推进，并且前后两个操作之间的数据、状态有一定的关系。由此产生了顺序程序的特点：

顺序性。一个程序的各操作在处理机上以严格的顺序执行。即下一个操作必须在上一个操作结束后才能运行。

封闭性。封闭性指程序在执行的过程中独占计算机系统的全部资源，不受外界因素的影响，系统的状态只有该程序可以改变。

可再现性。当程序重复执行时，如果初始数据相同，总是可以得到相同的结果，程序的结果与程序执行的速度无关。可再现性的特点为程序员检测和校正程序错误带来了极大的方便。

3.1.2　程序并发执行

1. 程序的并发执行

图 3-1 中，一个作业的计算过程分成 3 个操作完成，即输入、计算和打印结果，由于这 3 个操作是一个程序的逻辑顺序，因此，这 3 个操作必须顺序执行。为了增强系统硬件资源的利用率，在现代计算机系统中普遍采用同时性操作技术。从图 3-1 可以看到，3 个不同的操作分别在输入机、中央处理机和打印机上执行，这 3 个设备实际上是可以同时操作的，但是由于程序本身的逻辑性，这 3 个操作还是只能顺序执行。在顺序执行的过程中，可以发现，当其中的一个设备处于工作状态时，其他两个设备都处于空闲状态，导致了系统资源利用率不高。

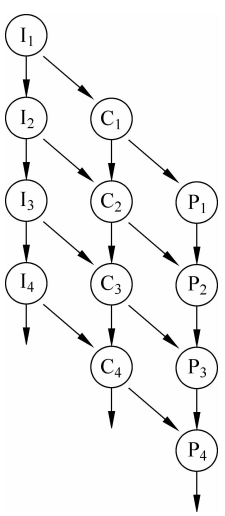

图 3-2　程序段并发执行

为了提高设备的利用率，当有一批作业需要处理时，可以让多个设备同时操作，如图 3-2 所示。图中给出了 4 个作业，每个作业都分为 3 个顺序操作：输入 I、计算 C 和结果打印 P，可以看到，I_3、C_2、P_1 分别属于不同作业的操作。在某个时刻，它们分别在使用输入、计算和打印设备，从时间上看，这 3 个设备都在工作，虽然它们在执行不同的操作。如果系统中有大量的作业，并

且作业都经过输入、计算和打印结果 3 个操作，则相应的 3 个设备几乎都处于满负荷工作状态。因此，设备得到了充分利用，系统的吞吐率也得到极大提高。

所谓程序的并发执行是指：若干个程序段同时在系统中运行，这些程序段的执行在时间上是重叠的，一个程序段尚未结束，另一个程序段已经开始执行，即使这种重叠是很小的一部分，也称这几个程序段是并发执行的。在描述并发执行的程序时，通常使用 Dijkstra 最先提出的方法，如图 3-2 所示，I_3、C_2、P_1 是并发执行的，则使用下面的语句描述：

```
cobegin
    I3;
    C2;
    P1;
coend;
```

cobegin 和 coend 两个关键字间的语句是并发执行的，并发执行的 3 个语句在书写顺序上并不表示执行的先后顺序。

2. 程序并发执行的特点

程序并发执行提高了资源利用率和系统处理能力，但是也带来了与顺序执行程序不同的新特性。

1）失去了程序的封闭性

在顺序执行程序的时候，系统所有的硬件资源和软件（如变量、表格等）资源由一个程序独享，程序在执行的过程中不受任何干扰，即这个程序执行后的输出结果是一个与时间无关的函数，具有封闭性。如果一个程序在执行的过程中可以改变另一个程序的变量，那么，后者的输出有可能依赖于各程序执行的相对顺序，也就是说，在并发执行程序的时候，程序失去了封闭性，多次执行时，由于执行程序的相对顺序不同，可能得到不同的结果。

```
begin
    int n = 0;
    cobegin
        while(A 任务没有完成){ //程序段 A
            ⋮
            n++;
        ⋮
        }
        while(A 任务没有完成){ //程序段 B
            ⋮
            printf("n = %d\n",n);
            n = 0;
            ⋮
        }
    coend
end
```

在上面的代码中，有两个程序段 A 和 B，A 每执行一次都要对变量 n 进行加 1 操作；B 每执行一次先打印出 n 的值，再将 n 置 0。由于 A 中的 n++操作既可以在程序段 B 的两个语句之前执行，也可以在中间和之后执行，对于这 3 种情况，打印的分别是 n++、n 和 n，执行后的值分别是 0、1、0。这种多个执行结果的错误原因在于：程序段 A 和 B 公用了一个变量

n，而程序段的执行也没有采取适当措施，这样，程序计算的结果就与时间有关，这种与时间有关的错误使程序失去了封闭性，结果便不可再现。

2）间断性

并发执行的程序之间有两种制约关系：

（1）间接关系，由于程序共享硬件和软件资源引起；

（2）直接关系，因程序相互合作共同完成一项任务而产生。

例如，两个程序共享打印机资源，一个程序正在使用时，另外一个程序就只能等待，这种共享导致第二个程序处于暂停状态，打印机可用后，该程序将继续执行。这种共享资源的关系使得程序以不可预知速度向前推进，换而言之，程序之间的相互制约关系导致程序“执行—暂停—执行”，即程序的执行具有间断性。

在第二种制约关系中，一个程序的若干个任务相互协作共同完成一个任务，有些任务可以并发执行，有些任务只有等到其他任务完成后才能执行，这也导致了程序执行的间断性。

3）通信性

对于相互合作的程序，为了更有效地协调运行，相互之间应进行通信，即一个程序向另外一个程序发送消息，以便接收到消息的程序可以启动执行。如图 3-3 所示，三个程序 A、B、C 相互协作共同完成一个任务，A 和 B 可以同时执行，而 C 只有在收到 A 和 B 都完成后的消息才能开始执行，这里，A 和 C、B 和 C 之间存在消息通信，虽然 A 和 B 发送的只是一个简单的完成消息，但足以让程序 C 开始执行。在合作程序之间有时也可能发送大量的消息。

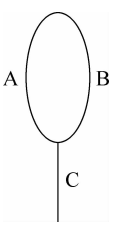

图 3-3　并发进程间的通信

4）独立性

在并发程序的运行过程中，各程序都是一个独立运行的实体，具有作为一个单位去申请资源的独立性，否则将无法运行。例如，一个程序要从其他程序接收大量消息，则该程序必须能够申请到相应的缓冲区，只有申请到缓冲，才可以进行接收操作。信息的到达是随机的，因此，接收程序随时都可能申请缓冲，使用完再立即释放。

3.1.3　进程描述

1. 进程定义

由于程序之间的直接和间接的制约关系，并发程序在执行时，与外界发生了密切的联系，使得程序总是在“执行—暂停—执行”这样有规律的状态之间变换，并发执行的程序因此具有间断性的特点，程序也因此失去了封闭性。

在这种情况下，如果仍然使用程序的概念，则只能对程序并发执行进行静止的、孤立的研究，不能深刻反映他们活动的规律和状态的变化。为此，人们引入了新的概念“进程”（process），以便从变化的角度，动态地分析并研究并发程序的活动。

进程的概念早在 20 世纪 60 年代初期，首先由麻省理工学院的 MULTICS 系统和 IBM 的 CTSS/360 引入。之后，许多专业人士从不同角度对进程进行了各式各样的定义。其中，最能反映进程实质的定义有：

（1）进程是程序的一次执行活动；

（2）进程是可以和别的计算并发执行的计算；

(3) 进程是一个程序在对应数据结构上的进行的操作；

(4) 所谓进程，就是一个程序在给定活动空间和初始环境下，在一个处理机上的执行过程；

(5) 进程是程序在一个数据集合上运行的过程，它是系统进程资源分配和调度的一个独立单位。

根据1978年在庐山召开的全国计算机操作系统会议上关于进程的讨论，结合国外的各种观点，国内关于进程的定义如下：

进程，是一个具有一定独立功能的程序，是关于某个数据集合的一次运行活动。这里给的进程定义，侧重于三点：进程是一个程序段；进程的操作对象是一个数据集合；进程是一个在处理机上的活动过程。早期关于进程的定义在本质上是相同的，只是侧重点不同。

进程和程序既有联系，又有区别，它们的区别和关系在于：

(1) 程序是指令的有序集合，其本身没有任何运行的含义，它是一个静态的概念。进程是程序在处理机上的一次执行过程，它是一个动态的概念。程序可以作为一种软件数据长期保存，而进程是有一定生命期的，它能够动态地产生和消亡。即进程可因“创建”而产生，因调度而执行，因得不到资源而暂停，以致最后因“撤销”而消亡。

(2) 进程具有并行特征，能与其他进程并行地活动；

(3) 进程是竞争计算机系统有限资源的单位，也是进行处理机调度的基本单位。

(4) 同一程序同时运行于若干不同的数据集合上，它将属于若干个不同的进程。或者说，若干不同的进程可以包含相同的程序。也就是说，用同一程序对不同的数据先后或同时加以处理，就对应于好几个进程。

(5) 进程是由程序段、数据段和进程控制块组成。它是程序段在某种数据段的执行。

2. 进程类型

系统中有很多进程，按照性质来分，有系统进程和用户进程。系统进程对资源进行管理、控制用户进程的执行；而用户进程主要是为用户完成计算任务。它们之间的区别在于：

(1) 系统进程被分配一个初始的资源集合，这些资源可以为系统进程独占，也可以按最高优先权限优先使用。用户进程要使用资源，必须通过系统服务请求来申请资源，并和其他用户进程竞争资源。

(2) 用户进程不能直接完成I/O操作，而系统进程可以做显示的、直接的I/O操作。

(3) 系统进程在管态下运行，而用户进程在目态下运行。

另外，还可以根据进程活动特点将其分为计算进程和I/O进程。计算进程在其活动期间主要在CPU上完成大量的计算工作，而I/O部分很少；I/O进程正好相反，其主要任务是使用I/O设备完成数据的输入/输出，而计算量很少。例如，科学计算任务往往要求大量的计算工作，占用了很多CPU时间，I/O任务很少，而数据查询检索正好相反，要求较少的计算，而要求大量的输入输出。

3. 进程的特征和利弊

进程具有下面的特征：

(1) 动态性。进程的定义描述了进程是程序的一次执行活动，因此，动态性是进程最基本的特征。进程因创建而产生，之后因调度而执行，因得不到资源而等待，因完成任务而消

亡。可见,进程的状态是动态变化的,相应的任务完成时,其生命期就结束。

(2) 并发性。引入进程概念的原因正是因为程序并发执行的目的。并发性是进程的第二特征。程序要并发执行,首先需要系统为之创建进程,只有进程才能并发执行,程序是静止的,不能并发执行。

(3) 独立性。进程是一个能独立运行的基本单位,同时也是系统分配资源的和调度的单位。进程在获得需要的资源后就可以开始运行,如果得不到资源便暂时停止,等待资源可用。未建立进程的程序是不能作为一个单位独立运行的。

(4) 异步性/间断性。进程按照不可预知的速度向前推进,进程执行的规律是“执行—暂停—执行”,其原因在于进程之间的直接和间接的制约关系。

(5) 结构特征。为了描述进程动态变化的过程,并使之能独立运行,每个进程都有一个数据结构,这个数据结构称为进程控制块 PCB,一方面 PCB 描述进程,如对应的程序代码、运行时需要的数据和堆栈,另一方面,PCB 也包含一些控制信息。

引入进程的原因是程序并发执行的需要,只有为每道程序建立了进程,它们才能并发执行。并发可以改善系统资源利用率和系统吞吐量,因此,目前在大、中、小型计算机,甚至在高档微机中,几乎都引入了进程,但是系统也必须为引入进程增加开销:

(1) 空间开销。引入进程后,系统要为每个进程建立一个数据结构,即进程控制块 PCB,它通常要占用几十到几百个内存字节;为了协调进程间的并发活动,系统还必须设置相应的管理机构,这也需要消耗可观的内存空间。

(2) 时间开销。系统需要花时间来控制并发活动的进程,比如,在处理机上切换进程,系统需要保存前一个进程的现场,恢复后一个进程的上一次执行的状态,这都需要 CPU 花时间来处理。

4. 进程的状态与变迁

进程并发执行的特征决定了进程总是处于“执行—暂停—执行”状态,直到进程完成任务并消亡。在进程的生命周期,进程有时处于运行,有时因为得不到要求的资源而处于暂停状态,当要求的资源可用后因为 CPU 忙而又处于准备就绪状态。所以,一个进程至少具有三种基本状态,即就绪、执行、等待状态。

(1) 就绪状态(ready)。一个进程申请到了所需要的除 CPU 以外的资源,具备了执行的条件,但是由于 CPU 正在处理其他进程而不能在 CPU 上运行,这样的进程所处的状态就是就绪状态。就绪状态的进程一旦得到 CPU,就可以开始运行。通常,在系统中处于就绪状态的进程有多个,它们排成一个队列,称为就绪状态。

(2) 执行状态(running)。系统的进程调度程序在处理机空闲时,从就绪队列中选择一个进程,并将处理机分配给它,进程得到 CPU 资源后,便可以在处理机上执行,这时的状态为执行状态。在单处理机系统中,只有一个进程处于执行状态,在多处理机系统中,可以有多个进程处于执行状态。

(3) 等待状态(wait)/阻塞状态。正在执行的进程,因为等待某一事件发生(如等待输入输出操作完成)放弃处理机而暂时停止执行,这时,进程的状态就是等待状态。引起进程进入等待状态的典型事件有请求 I/O、申请缓冲空间等。当等待的事件发生或完成后,进程将进入就绪队列,等待被进程调度模块选中进行下一次运行。

应当指出,当正在执行的进程进入等待状态后,系统的进程调度模块立即将处理机分配

给另一个就绪进程；当一个进程等待的条件发生后，该进程将由等待状态变为就绪状态重新等待获得处理，而不是直接恢复到执行状态。可见，进程的状态是不断变化的，进程的3种基本状态及其状态变迁如图3-4所示。

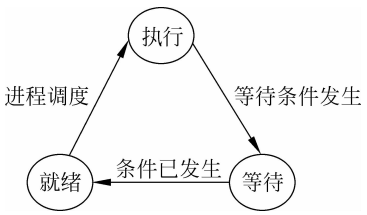

图3-4 进程基本状态变迁

上面介绍了进程的三种基本状态和其变迁，那么，进程是如何产生和消亡的呢？进程是程序的一次执行过程，是一个活动。当系统或用户需要一个活动时，可以通过创建进程的方法来产生一个进程，被创建的进程直接进入就绪状态。当一个进程的任务完成，系统可以通过撤销进程的方法使得进程消亡。

图3-4描写进程的基本状态，在不同的系统中，可以设置更多的状态，其变迁将更复杂。比如，在有的系统中，有时希望将正在执行的进程或者没有执行的进程挂起(suspend)，使之处于静止状态，以便研究该进程的执行情况或者对它进行修改，这样的进程称为挂起状态；在分时系统中，正在执行的进程因为时间片到可以直接变为就绪状态；在Linux系统中，就绪/等待状态也可以分为内存就绪/等待状态和外存就绪/等待状态，其状态变迁将更复杂。

5. 进程描述——进程控制块PCB

1) 进程控制块的作用

引入进程概念后，系统为了描述和控制进程的运行，刻画进程在不同时期所处的状态，为每个进程定义了一个数据结构，即进程控制块(Process Control Block，PCB)。系统根据PCB感知进程的存在，因此，PCB是标识进程存在的唯一实体。当系统为用户程序创建一个进程时，实际上是为该进程分配一个PCB，并用相应的数据填充，在该进程的整个生命周期，系统使用这个PCB中的信息对该进程实施控制和管理。进程任务完成，系统回收其PCB，该进程便消亡。在一个实际的系统中，PCB的数量通常是一定的(比如200)，该数字规定了系统所允许的最多进程数。系统将所有PCB形成一个结构数组，放在操作系统专用区。

2) 进程控制块的内容

进程控制块PCB是一个数据结构，包含了进程的描述信息和控制信息，常用的信息见表3-1。

表3-1 PCB信息

进程标识符	进程通信信息
进程当前状态	互斥和同步机构
现场保护区	家族联系
程序和数据地址	链接字
进程优先级	总链指针
资源清单	

(1) 进程标识符。每个进程都必须有一个唯一的标识符，它是一个整数，也称进程内部名称。在创建进程时，系统为之分配一个空白PCB，并分配唯一标识符。

(2) 进程当前状态。说明本进程目前的状态(就绪、执行、等待等)，作为进程调度程序分配处理机的主要依据。只有当进程处于就绪状态时，才有可能分配到处理机；当进程处于等待状态时，要在PCB中说明等待的原因。

(3) 现场保护区。当进程由于某种原因需要从执行状态变为等待状态时,其 CPU 现场信息必须保存,以便将来继续运行。要保存的信息包括:程序状态字、通用寄存器、指令计数器等。

(4) 程序和数据地址。从进程的定义可以看到,进程是一段程序对一个数据集合的操作过程。因此,在 PCB 中要说明该进程对应的程序和数据的地址,以便处理机执行时能找到指令和数据。

(5) 进程优先级。优先级反映了进程要求 CPU 的紧迫程度,通常,优先级是一个整数,由用户预先提出或由系统指定,优先级高的进程可以优先获得处理机。

(6) 资源清单。它列出了进程所需要的资源以及当前已经分配到的资源种类和数量。

(7) 进程通信信息。用于实现进程之间的通信所需要的数据结构,如指向信箱或消息队列的指针等。

(8) 互斥和同步机构。实现进程之间互斥和同步所需的机构,如信号量和锁等。互斥和同步是进程之间的两种制约关系。

(9) 家族联系。系统为用户程序创建进程,进程又可以创建子进程,依次类推,进程之间可以形成一个家族关系。家族联系说明了本进程与其家族进程之间的关系。

(10) 链接字。指出了和本进程处于同一状态的下一个进程的指针。

(11) 总链指针。系统中有大量的进程,并且处于不同的状态,所有的进程在一个队列,形成一个链表,当前状态字区分不同进程的状态。总链指针即头指针。

3) PCB 的组织方式

一个系统中的 PCB 有很多,为了对它们进程有效管理,必须用适当的方式将它们组织起来,目前常用的组织方式有以下几种:

(1) 链接方式。如图 3-5 所示,左边是 4 个首指针,分别指向处于不同状态的进程队列,这样,系统就可以形成就绪队列、等待队列、空闲队列和执行队列。在单处理机系统中,执行队列最多只能有一个 PCB 结点。就绪队列中的 PCB 结点可以根据优先级排列;等待队列中的 PCB 结点可以根据等待原因排列。

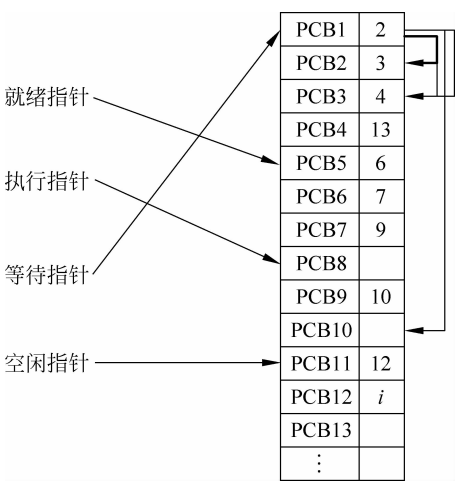

图 3-5 按链接方式组织 PCB

(2) 索引方式。系统可以根据进程的状态建立几张索引表,如图 3-6 所示,有就绪索引表、等待索引表和空闲 PCB 索引表,并将各索引表的首地址放在专用指针中。在每个索引

表的表项中，记录具有相应状态的某个 PCB 在 PCB 表中的地址。

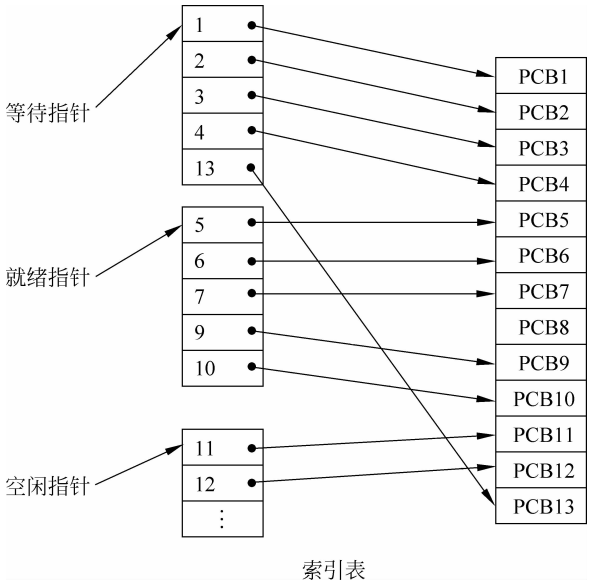

图 3-6 按索引方式组织 PCB

3.2 进程管理

并发系统中有许多进程，为了实现进程管理、协调并发进程之间运行，系统就必须提供对进程实行有效管理和控制的功能。进程控制的任务就是对系统中的所有进程实施有效管理，它是处理机管理的一部分。进程控制模块是操作系统内核的重要功能之一，它具有进程的创建、撤销，进程状态转换和实施进程间同步、通信等功能，这些功能往往是通过一组原语操作来实现。所谓原语，是一种机器指令的延伸，是由若干条机器指令构成用以完成特定功能的一段程序，一般为外层软件所调用。其特点是具有原子性，即原语操作是一个不可分割的操作，要么全部做，要么全部不做。在操作系统中，原语是作为一个基本单位出现的。用于进程控制的原语有创建原语、撤销原语、等待原语和唤醒原语等。

3.2.1 进程创建原语

在系统初启成功时，系统已经生成了一些必需的、承担系统资源分配和管理工作的进程，称为系统进程。当用户的作业进入系统，由操作系统的作业调度这一系统进程为它创建相应的若干用户进程。在层次结构的系统中，允许一个进程创建一些附属进程，以完成一些可以并行的工作，这时的创建者称为父进程，被创建者称为子进程，创建父进程的进程称为祖先进程，这样就构成了一个进程家族。用户不能直接创建进程，而只能通过系统请求的方式向操作系统申请创建进程。为用户作业创建新进程是进程管理的基本功能之一。

无论是系统还是用户创建进程，都必须调用进程创建原语来实现。进程创建原语的主要功能是为被创建进程建立一个 PCB，因此，调用创建原语的进程必须提供 PCB 的有关参数，以便填入相应的 PCB 中。填入的主要参数有新进程的符号名，优先级，开始执行地址等。其他参数可以从父进程那里直接继承。

创建进程原语的操作过程是：首先从空闲 PCB 队列中申请一个可用的 PCB，申请到后

为该 PCB 分配一个内部标识符；然后填入创建者提供的参数和直接从父进程继承的参数；把新进程设为就绪状态，并插入到就绪队列和进程家族；最后，返回为新进程的内部标识 PID。创建原语描述如下：

```
算法：create
输入：新进程的符号名，优先级，开始执行地址
输出：新创建进程的内部标识符 PID
{
在总链队列上查找有无同名的进程；
if(有同名进程) return(错误码)          /*带错误码返回*/
在空闲 PCB 队列申请一个空闲的 PCB 结构；
if(无空 PCB 结构) return(错误码)；      /*带错误码返回*/
用参数填充 PCB 内容；
置进程为就绪状态；
将新进程的 PCB 插入到就绪队列；
将新进程的 PCB 插入到总链队列中；
设置进程的家族关系；
return(新进程 PID)；
}    /* create(name,priority,start-addr)   */
```

返回参数 PID 是操作系统给出的唯一标识进程存在的符号，它是一个数值型的数据，以后操作系统对该进程的操作就以 PID 为基准。要创建新进程就必须调用系统的进程创建原语，该原语被调度的时机通常是：

(1) 系统启动时调用创建原语创建系统进程；

(2) 用户作业进入系统时由作业调度进程为作业创建若干进程；

(3) 用户用显示的调用语句使用该原语创建进程。

3.2.2 进程撤销原语

一个进程完成任务后应予以撤销，以便及时释放其所占用的各种资源。这时应使用进程撤销原语 Kill 撤销进程。撤销原语根据调用者提供的进程标识符去检索想要撤销的进程 PCB，获得该进程的内部标识和状态。如果是正在执行的进程，则立即停止该进程运行，将 PCB 归还给系统，将所占的资源还给父进程，并从总队列中摘除它，再转进程调度程序，使得下一个合适的进程可以运行，撤销原语 Kill 操作如下：

```
void kill
输入：进程标识符 PID
输出：无
{
  由参数 PID 查找到当前进程的 PCB；
  释放本进程所占用的资源给父进程；
  将该进程从总链队列中摘除；
  释放此 PCB 结构；
  释放所占用的资源；
  转进程调度程序；
} /*  kill  */
```

说明：进程有家族关系，如果被撤销的进程有子孙进程，其子孙进程将全部撤销，这时递归调用 Kill 即可；在释放资源时，如果是被撤销进程自己申请的资源，则还给系统，如果是继承的父进程的资源，则还给父进程。进程撤销原语被调度的典型时机有：

(1) 一个进程运行完毕，调用该原语撤销自己；

(2) 一个进程可以调用该原语撤销其子孙进程；

(3) 系统进程调用该原语撤销用户进程。

3.2.3 进程等待原语

当进程需要等待某一事件完成时，它可以调用等待或者阻塞原语把自己挂起，使自己从就绪状态变为等待状态。进程一旦被挂起，它只能由另一个进程唤醒，而不能自己叫醒自己。等待原语操作过程如下：

```
void susp(chan)
输入：chan /*等待的事件(等待原因)*/
{
  保护现行进程 CPU 现场到 PCB 结构中；
  置该进程为"等待/阻塞"态；
  将该进程 PCB 插入到等 chan 的等待队列；
  转进程调度；
}  /*  susp(chan)  */
```

该原语的功能是，停止调用该原语的进程的执行；将 CPU 现场保存在该进程的 PCB 结构中；将其状态改为等待状态并加入到 chan 的等待队列；之后是控制转向进程调度程序，选择另外一个进程在 CPU 上运行。这里，处于等待状态的进程很多，可以按照等待的原因将处于等待的进程放在不同的等待队列。一般情况下，是进程自己调用等待原语将自己放在相应的等待队列。

3.2.4 进程唤醒原语

当进程等待的事件发生时，调用唤醒原语将等待该事件的所有进程唤醒，使得它们有机会继续执行。进程处于等待状态时，不能自己唤醒自己，只能由其他进程，或者发现者进程，或者系统监控进程唤醒它。比如，一进程在等待打印机，当正在使用打印机的进程操作完毕，这一进程将发现还有进程在等待打印机，便调用唤醒原语唤醒这一等待打印机的进程使用打印机。另一例子，是相互合作的进程之间的唤醒，如图 3-3 所示，进程 A 或者 B 完成后唤醒进程 C 执行。唤醒原语操作过程如下：

```
void wakeup(chan)
输入：chan /*等待的事件(等待原因)*/
输出：无
{
   保护现行运行进程的 CPU 现场到它的 PCB 结构中；
   置该进程为就绪状态；
   将该进程插入就绪队列；
   找到该阻塞原因的队列指针；
```