微型计算机实用大

第8篇 操作系统篇

8.1 操作系统基本概念与原理

8.1.1 操作系统的基本概念

操作系统 操作系统是直接控制和管理计算机硬件和软件资源，以方便用户，充分有效地利用这些资源的程序集合。操作系统是最基本的系统软件，它密切地依赖于计算机硬件，直接管理着系统中的各种硬件和软件资源。

操作系统的对外界面称做系统调用。实用层软件及各种应用软件均是通过系统调用访问计算机的软硬件资源的。所谓系统调用，实际上是操作系统软件提供的，能访问操作系统核心的接口程序。

一个好的操作系统应提供给用户一个清晰、简洁、易于使用的用户界面。这里的"用户"是个广义的概念，他既是计算机系统的最终用户，又包括计算机系统的管理员，甚至包括实用层程序的设计人员。其次，操作系统应尽可能地使系统中各种资源得到最充分的利用，这里的"资源"包括CPU、内存、外存和I/O设备等硬件资源，以及各种软件资源。

操作系统这一术语有着各种定义和说明，下面是几个有代表性的定义：

(1)一般地说，操作系统被定义为管理其他程序运行的程序；

(2)所谓操作系统，指的是用以控制和管理硬件和软件资源以及方便用户的程序的集合。

操作系统是计算机系统中的重要组成部分，它有效地统管计算机的所有资源，合理地组织计算机的整个工作流程，以提高资源的利用率，并为用户提供强有力的使用功能和灵活方便的使用环境。

系统的资源分为四类：处理机、存储器、I/O设备和信息，相应的操作系统就应包括这样几个部分：

(1)用于控制和管理处理机的程序；

(2)控制和管理存贮器的程序；

(3)控制和管理I/O设备的程序；

(4)控制和管理程序和数据的程序。

由此可见，操作系统是计算机系统中极其重要而又基本的系统软件之一。

操作系统的组成 操作系统一般由5个部分组成：进程管理、存储管理、文件管理、设备管理和作业管理。

1. 进程管理

进程管理又叫处理机管理。在硬件系统中，CPU是极其宝贵的资源。现代的计算机系统虽然广泛采用了多处理机技术，但使用最多的还是集中式的系统，即系统中只有一个功能强大的主CPU，并由它完成主要的处理任务。进程管理是通过引入"进程"的概念使CPU资源得到最充分的利用。

进程管理的第一部分工作是处理中断事件，硬件处理机只能发现中断事件并产生中断，但不能处理，配

置了操作系统之后就能对各种事件进行处理。进程管理的第二部分工作是处理机的调度，处理机可能是一个或多个，不同的操作系统将针对各种不同情况而采用不同的调度策略。

2. 存储管理

存储管理主要管理内存资源。内存是计算机硬件中除CPU之外的另一个宝贵资源，虽然内存的价格在不断地下降，但它在整个系统中仍是较昂贵的资源。存储管理对内存资源进行统一管理，使多个用户能够分享有限的内存资源，以及存储在内存中的数据和程序。此外，存储管理还负责对用户自己的程序和数据进行保护，防止它们受到破坏，从而使各个用户互不干扰地使用同一个内存。操作系统的部分功能与硬件存储器的组织结构密切相关，应根据硬件情况和使用需要，采用各种相应的有效调度策略与保护措施。

3. 文件管理

计算机内存的容量有限，成本高，因此大部分用户程序的数据、实有程序，甚至操作系统本身常常放在外存储器(磁盘、磁带)上，对于这样复杂、庞大的软件资源必须有条理地进行组织、存储、保护，以使用户能方便、安全地访问它们，这项工作就是由文件管理来完成的。

文件管理支持对文件的存储、检索和修改等操作以及文件保护的功能。早期的管理程序仅提供一个简单的文件系统。操作系统一般都提供功能较强的文件系统，许多系统还提供数据库系统来实现信息的管理工作。

4. 设备管理

现代的计算机系统能支持各种各样的设备，这些设备之间往往关系复杂，功能特性也各不相同。设备管理就是对这些设备进行统一管理，充分发挥它们的效率并使用户以方便、统一的方法使用这些设备。

设备管理负责管理各类外围设备，包括分配、启动和故障处理等。为了提高效率，还引入了逻辑(虚拟)设备的概念，以实现预输入和缓输出的功能。

5. 作业管理

作业是指用户程序、数据，以及运行这些程序和处理这些数据过程中，用户的各种要求。作业管理的功能是提供给用户一个使用计算机系统的界面，使用户能够方便地运行自己的作业，并对进入系统的所有用户作业进行管理和组织，以便提高整个系统的运行效率。

操作系统的类型 操作系统有许多分类方法。比如，按用户数目的多少，可把操作系统分为单用户和多用户系统。又如，根据操作系统所依赖硬件的规模，可把它们分成大型机、中型机、小型机和微型机操作系统。虽然分类的方法很多，但最常用的一种分类方法是按照提供的功能进行分类，据此，操作系统大致可分成以下6类。

1. 单用户操作系统

单用户操作系统的主要特征是在一个计算机系统内一次只能支持运行一个用户程序。多数微型机操作系统都属于这样的系统，例如，CP/M，MS－DOS等。

2. 批处理操作系统

用户把要计算的问题、数据和作业说明书一起交给操作员，操作员将一批算题输入到计算机，然后由操作系统来控制执行。通常，采用这种批量处理作业技术的操作系统称为批处理操作系统。

批处理操作系统分为单道批处理系统和多道批处理系统。

(1)单道批处理系统。单道批处理系统是一种早期的、基本的批处理操作系统，它把程序设计语言、I/O支持和实用程序组成一个整体来控制和管理作业的运行。单道指的是一次只有一个作业进入计算机系统的内存运行。这种系统的主要目标是使整个作业流能自动、顺序地运行，以节省人工操作时间，提高效率。

(2)多道批处理系统。第三代计算机为操作系统提供了更充分的硬件支持，特别是磁盘的使用以及诸如系统异步控制、中断、特权与保护等概念加入硬件结构，促进了操作系统的发展。

一般地说，一个批处理多道系统的主要特征有：①内存管理。如允许程序在运行期间动态地获得或释放内存。②共享代码的能力。一些例行子程序可以常驻内存，并为多个程序共享。

3. 实时操作系统

在计算机的某些应用领域内，要求对实时采样数据进行及时处理并作出相应的反映，如果超过限定时间就可能丢失信息或影响到下一批信息的处理。为了满足这种实时响应的要求引入了实时系统。实时系统有三种典型应用形式，这就是过程控制系统、信息查询系统和事务处理系统。

4. 分时操作系统

分时操作系统使计算机为一组终端用户服务，使得每个用户好像自己有一台支持自己请求服务的计算机。它把CPU分成一个一个的时间片，并把每个时间片分给各个并发程序，每道程序一次只运行一个时间片，当时间片计数到时后，操作系统选择另一道程序并分给它时间片，让其投入运行，如此反复。因为时间片很短，使得系统上的各个用户都认为整个系统只为他自己服务。

5. 网络操作系统

提供网络通信和网络资源共享功能的操作系统称为网络操作系统。

6. 分布式操作系统

用于管理分布式系统资源的操作系统称为分布式操作系统。它与集中式操作系统的主要区别在于资源管理、进程通信和系统结构。

单道批处理系统 通常，一个用户程序是有着一定独立性的程序段的有序集合。用户程序及其所需的数据和命令一起形成一个作业；逻辑上作业是由有序的作业步组成，每个作业步又由完成作业中某一相对独立事件的程序和数据构成，并由命令定义之。在批处理环境中，通常都是把一批作业有序地排在一起形成一个作业流，这同样也采取命令形式来标识一个新作业的开始和前一作业的结束。

在50年代，计算机的工作基本上采用人工操作，它将纸带装进纸带输入机，把程序和数据输入，然后通过控制台开关启动程序。这种人工操作方式具有如下特点：①用户独占全机；②CPU等待人工操作；以上两点说明人工操作严重地损害了资源的利用率，此即所谓的人机矛盾。

在脱机输入输出方式中，由于事先已把若干个作业记录在一盘磁带上，这意味着用户程序的处理顺序已经排定。机器将首先处理磁带上的第一个作业，即把它由磁带传送至内存后启动它，并把控制机交给作业1。当作业1完成后又把控制权还给系统，系统再把磁带上的下一个作业输入内存并启动它，把控制权交给第二个作业。按这种方式对磁带上的作业自动地一个接一个地进行处理，于是便形成了批量处理系统。不难看出，单道批量处理系统是在解决人机矛盾和CPU－I/O速度矛盾的过程中，也就是在提高资源利用率的过程中发展起来的。此时，为对系统中的作业和计算机各种资源进行监督和管理，又相应的配置了监督程序或管理程序，它不仅是操作系统的前身，而且也是操作系统的核心部分。

为使监督程序能代替操作员对作业进行控制和管理，用户必须通过某种命令方式，把与作业有关的信息提供给系统。为实现这些功能，系统必须配置相应的命令解释程序。在批处理环境中，系统要能识别一个作业步的开始和结果，以便当一个作业或作业步完成时，能自动过渡到下一个作业，这需要一个作业定序程序来完成，命令解释程序和定序程序两部分组成了监督程序中的作业控制部分。监督程序具有内存管理和设备管理的功能。

多道批处理系统 虽然单道批处理系统已大大减少了人工操作的时间，提高了机器的利用率，但是对于某些作业在发出I/O请求后，还必须等待，而在单道程序运行中的作业，等待就意味着机器空闲。特别是由于I/O设备的低速性，将导致机器的利用率非常低。为了改善CPU的利用率而引入多道程序，即同时把几个作业放入内存，它们分时共用一台计算机。CPU先对第一道作业进行处理，当它需要I/O时，CPU处理它的I/O请求后便转向第二道程序，使第一道程序的I/O操作和第二道程序的处理并行。

多道程序系统是利用CPU等待时间来运行其它程序，这就显著地提高了资源的利用率，从而增强了系统的吞吐能力。

把批处理系统同多道程序系统相结合，就形成了多道批处理系统，这种系统从60年代初出现以来，目前还在广泛使用。

多道批处理系统的运行方式是:在外存中存放大量的后备作业,系统根据一定的调度原则从后备作业中选择搭配合理的一批作业,调入内存中以多道程序的方式运行。所谓合理搭配,主要是指作业的选择既应有利于提高系统资源的利用率,又能满足不同用户的响应时间要求。

在多道批处理系统中管理程序按照系统资源的类型可分为4种:

(1)存储器管理。按一定策略将内存分配给多道程序,使它们不致因相互重叠而丢失信息;此外,还必须防止因某道程序出现异常情况而破坏其他程序;第三点是必须具有内存扩充能力,使大的用户程序可以在这样的系统中运行。

(2)处理机管理。在多道批处理机系统中,处理机为几道程序共享,因此处理机管理的基本功能是按照一定的策略把处理机分配给某个程序,使之运行一指定时间,然后再分配给另一用户程序,这样几道程序便可有条不紊地在一个系统中运行。

(3)I/O设备管理。按照设备类型和一定策略把I/O设备分给某些作业,同时还分配相应的通道和控制器。

(4)文件系统。把大量有意义的信息以文件形式存放在各种存储器中,以提供给所有的或指定用户使用。

在单处理机系统中运行多道程序后,每个用户都认为系统为自己提供一台处理机,这样的处理机是逻辑上的,称之为虚处理机。为建立多台虚处理机,应采用下述3点措施:

(1)分时使用处理机;

(2)设置管理程序;

(3)设置进程控制块PCB等数据结构。

分时系统 在批处理系统的基础上,引入远地作业进入方式和人-机交互作用而形成了分时系统。

在分时系统中,分时主要是指若干并发程序对CPU时间的共享,它是通过操作系统软件实现的,分享的时间单位称为时间片,它往往是很短的,如几十毫秒。这种分时的实现,需要有中断机构的时钟系统的支持。利用时钟系统把CPU分成一个一个的时间片,操作系统轮流把每个时间片分给各个并发程序,每道程序一次只运行一个时间片,当时间片计数到时后,产生一个时钟中断,控制转向操作系统。操作系统选择另一道程序并分给它时间片,让其投入运行,如此反复。因为时间片很短,使得系统上的各个用户都认为整个系统只为他自己服务。

分时系统的基本特征为:

(1)同时性:若干个用户能同时或基本上同时使用计算机系统;

(2)独立性:用户可以彼此独立地操作;

(3)及时性:用户能在很短的时间内获得对系统所提要求的回答;

(4)交互作用性:用户能与系统进行人-机对话。由于交互作用是分时系统的重要属性,因而分时系统又被称为交互作用系统。

分时系统具有如下优点:

(1)促进了计算机的普遍应用。

(2)节省开支、减少维护人员。

(3)可作为工程设计和方案论证的得力工具。

(4)显著地提高了研究、检查和调整程序的效率。

(5)进一步充分利用系统资源。

实时系统 实时控制系统和实时处理系统统称为实时系统。所谓实时,表示"立即"、"现在",而实时系统则是指系统对特定输入作出反应所具有的速度,用以控制发出实时信号的那个设备。

虽然多道程序系统和分时系统,已能获得令人满意的机器利用率和响应时间,使计算机的应用范围日益扩大,但它们仍不能满足某些领域对实时存取数据进行及时处理的要求,如:

(1)对飞机飞行、导弹发射等的实时控制;

(2)办理飞机票的预订、查询等实时信息处理。

实时系统和分时系统虽然都具有同时性、独立性、交互性以及及时性,但它们仍有重要区别:一方面,分时系统的目标是提供一种随时可供多个用户使用的、通用性很强的计算机系统,用户与系统之间具有较强的交互作用或会话能力;而实时系统则大都是具有特殊用途的专用系统,它仅允许终端操作员访问有限数量的专用程序。与分时系统相比,实时系统的交互作用能力较差。另一方面,分时系统虽对响应时间也有要求,但一般是以人所能接受的等待时间来确定,数量级为秒。而实时系统对响应时间的要求则是以控制过程或信息处理过程所能接受的延迟来确定的,数量级为秒,甚至微秒级。

实时系统大都带有专用性,种类繁多,规模的大小也相差甚大,因此各种实时系统的特性和功能必然有所不同。对于中、大型实时系统,存储器管理、处理机管理、I/O 设备管理以及文件系统是必须具备的基本功能,此外还应具有如下特性和功能:

(1)实时时钟管理——提供对实时任务进行时间管理能力和实时处理的能力。

(2)连续人机对话——终端发送一消息并收到计算机的回答后,终端又发送关于该问题的补充消息,这就要求计算机记住终端上次发来的消息,并根据本次接到的消息形成第二次回答。

(3)过载的防护——系统必须具有某种防护机构,以保证系统即使出现过载任务仍能正常运行。

(4)高可靠性——实时系统最重要的设计目标之一。必须采取相应的硬件和软件措施,提高系统的可靠性。

8.1.2 并发进程

进程的概念 进程(Process)用来描述系统中的各种并发活动,并为操作系统的设计提供一种清晰明确的方法。其定义为:可以和其他程序并发执行的一次程序执行。其特性如下:

(1)进程是一个动态的概念,它是一个过程。当系统要完成某项工作时,它应"创建"一个进程,进程的推进过程也就是工作的进行过程。当进程完成预定的任务之后,系统就要"撤消"这个进程。从进程"创建"到进程"撤消"这一全过程叫做进程的生存期。

(2)进程具有并发特性。在一个系统中,可能同时存在多个进程,这些进程可共同在系统中运行,轮流地占用CPU 和各种资源。

(3)进程之间相对独立、异步地运行。以分时系统为例,各个终端的用户进程是为了完成它们各自的任务,因此,这些进程在逻辑上是相互独立的,各个进程可以按不同的速度前进。

(4)进程需要程序、数据和描述其状态变化的实体。虽然进程是个抽象的概念,但它在系统中是实实在在存在的。进程要完成某一任务,就必须执行描述其任务所对应的程序,并对要求加工的数据进行操作。另外,由于进程在其生命周期之内可能申请和释放CPU 及各种资源,因此,它就会有状态的变化,为了记录进程的这种状态变化,系统就要为之设立专门的数据结构,即进程控制块。

进程的特性可归纳为动态性、并发性、独立性、异步性和结构性。

进程与程序有本质的不同。程序是静态的,它是指令的集合;而进程是动态的,它是程序执行的过程。一个程序可在系统上执行几遍,虽然都是执行同一个程序,但每次都对应一个不同的进程,即一个程序可以对应于多个进程。另外,多个进程同时共享内存中一个程序复本是一个程序对应多个进程的另一种情况。反之,一个进程也可以对应于多个程序,这时,程序的代码是由几个顺序执行的程序段所组成。

引入进程的概念后,多道程序的概念就可以用进程进行描述,即每一道程序可以看作是系统中的一个进程。另一方面,利用进程概念可以更清晰地描述系统中的各种并发活动,这使操作系统更容易设计、调试和维护。

进程的状态 进程有着"运行——暂停——运行"的活动规律，它并非一直处于执行状态。事实上，在运动中的进程至少具有3个基本状态：

(1)运行态(running)。当一个进程正在占用CPU时，它处于运行态。系统中处于运行态的进程不会大于CPU的数目。特别地，在单CPU的系统中，最多只能有一个进程处于运行状态。

(2)封锁态(blocked)。或称为阻塞态。当一个进程因等待某一个条件而不能运行时，即处于封锁状态。例如，一个进程申请某一设备，而该设备正在被其它进程使用，则该进程不能继续运行。处于封锁状态的进程在逻辑上是不可运行的。即使CPU空闲，它也不能占用CPU。

(3)就绪态(ready)。处于就绪态的进程在逻辑上是可运行的。但由于系统中CPU正在被其它进程占用，所以它在实际上(物理上)没能运行。一旦其他进程放弃了CPU，使CPU空出来，就绪进程就可以变为运行状态。

这三个基本状态隐含了这样的假定：所有进程都早已在系统中存在，并且按照"运行——阻塞——就绪——运行"这样的变化规律永无休止地循环下去。事实上，进程一般都有自己的发生、发展和消亡过程。一个更为完善的、更实际的状态图中还需增加三个状态。

(1)提交状态(submit)。用户通过输入机把作业提交给系统，系统必须响应用户的要求。

(2)收容状态(hold)。系统输入作业，并把它转换成机器内部形式，暂存在后援存储器上。但该进程尚未分配到必要的资源，只有在调度程序把内存及有关外部设备分配给它时，它才能由收容状态转换为就绪状态。

(3)完成状态。进程已完成自己的计算，系统收回分配给它的全部资源。

进程的状态转换 进程并非固定处于某个状态，它将随着自身的推进和外界条件变化而变化。

进程从封锁态到运行态必须经过就绪态而不可能直接转换到运行态。这是因为系统中可能有多个进程处于逻辑上可运行状态，因此系统必须根据各个进程任务的轻重缓急选择一个就绪进程占用CPU，这一选择过程称为进程调度(dispatch)。进程调度程序是操作系统中的进程管理软件的核心。

从进程状态的转换过程中，应注意到如下两点：

(1)一个进程由运行状态转换为封锁状态，一般是由运行进程自己主动提出的。例如，当进程在运行过程中，要求的某一条件得不到满足时，它就要主动地放弃CPU，而转为封锁状态，这一动作称为挂起(hang up)。

(2)一个进程由封锁状态变为就绪状态总是由外界事件所引起，而不是由该进程自己的运行所引起的。有时，称这一动作为唤醒(wake up)。例如，当某一个I/O操作完成时，I/O中断处理程序就把因为等待这一个I/O而挂起的进程唤醒。

在实际的操作系统中，进程的状态可能不止这三种，六种状态的转换图如图8.1-1所示。

在有些系统中为了调度上的方便，把三个基本状态进一步细化。如把就绪状态细分为低优先数就绪、中优先数就绪和高优先数就绪；把封锁状态细分为因等待盘或带I/O而封锁、因等待终端I/O而封锁、因等待页面I/O而封锁。

把正在执行或没有执行的进程挂起(Hang up)，使它们处于静止状态，以便研究它的执行情况或对它进行修改，因此增加一个挂起状态。在增加该状态后，一个进程或者是活动的(Active)，或者被挂起。把未被挂起的就绪状态和阻塞状态分别称为活动就绪(Readya)、活动阻塞(Blockeda)；而被挂起的就绪和阻塞状态分别称为静止就绪(Readys)和静止阻塞(Blockeds)。图8.1-2示出具有挂起状态的进程演变过程。

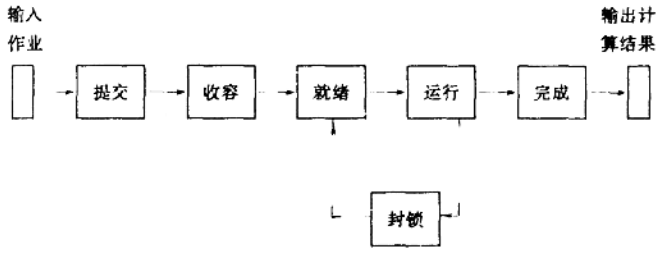

图8.1-1 六种进程状态转换图

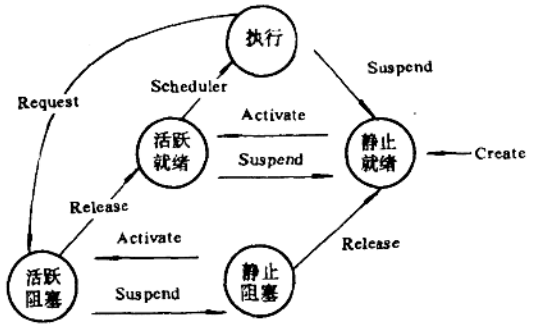

图8.1-2 具有挂起状态的进程演变图

进程控制块PCB 为描述进程的运动变化过程，采用了一个与进程相联系的进程控制块PCB(Process Control Block)或称进程描述器。它是进程映象中最关键的部分，含有进程的描述信息的控制信息，是特性的集中反映，是系统对进程施行识别和控制的依据。系统根据PCB而感知进程的存在，故PCB是进程的唯一实体。当系统创建一个进程时，必须为它设置一个PCB，再根据它对进程进行控制和管理。进程完成时，系统收回它的PCB，进程也随之消亡。为了对进程作充分的描述，PCB通常具有如下的信息：

(1)标识符(Identification)。进程的标识符是以字母或数字形式表示的进程名称。每个进程只能有唯一的标识符，不同的进程不能用相同的名字。在创建一个进程时，创建者必须给出该进程的标识符。在有的系统中，为了方便起见，每个进程具有两个标识符，一个是在创建时，由系统提供的内部名称，一个是由创建者给定的外部名称。

(2)进程当前状态(Status)。说明进程目前处于何种状态，作为进程调度时分配处理机的主要依据。只有当进程处于活跃就绪状态，且具有最高优先数时才可分得处理机。当某个进程处于阻塞状态时，要在PCB中说明阻塞的原因。

(3)CPU状态保护区。当进程由于某个事件而从执行状态变为阻塞状态时，CPU现场信息被保存在PCB的一个区域中，以便在重新获得处理机后能继续执行。通常被保护的信息有：工作寄存器和指令计数器中的信息或程序状态字等。

(4)进程起始地址。该进程将要从此地址开始执行。在采用上下界寄存器的内存管理中，还应给出上下界值。在页面管理中应给出页面表指针值。

(5)资源清单。每个程序段在运行时，除了需要内存外，还需要其它资源，如I/O设备、外存、数据区等。在该清单中将列出有关细目，例如磁盘存储器的片、通道、段的地址。

(6)进程优先数。在多进程系统中,由于进程数多于处理机数,致使各进程互相争夺CPU。一般根据进程的轻、重、缓、急规定一个优先数,进程调度程序根据它们优先数的高低进行调度,把CPU控制交给优先数最高的进程。

(7)队列指针(Pointer)或链接字(Link)。它用于将处于同一状态的进程链接成一个队列,在该单元中存放着本队列后一进程的PCB首址。

(8)家族联系。它说明本进程与本家族的关系,可设置指向父进程PCB的指针,指向第一子进程PCB的指针。

为了提高调度效率和便于对进程进行控制,PCB必须放在内存的系统区中。所有PCB按照一定方式组织起来,称为PCB表。PCB表是系统中最关键、最常用的数据,因为它的物理组织方式直接影响到系统的效率。PCB表的物理组织方式有若干种,但有两种常用方式:线性表和链接表。线性表的方式简单,最容易实现。操作系统预先确定整个系统中同时存在的进程的最大数目,静态分配空间,并把所有PCB放在这个表中,这种方式的主要问题是限定了系统中同时存在进程的最大数目。链接表是常用的方式,它按照进程的不同状态分别放在不同的队列中。链接表没有限制进程的数目,使用灵活、管理方便、内存使用效率很高。

进程操作原语 原语(Primitive)是机器指令的延伸,是由若干条机器指令构成的,用以完成特定功能的一段程序。为保证操作的正确性,原语在执行期间是不可分割的。在许多机器中为了实现上的方便,规定在执行原语操作时,要屏蔽中断,以保证原语操作的不可分割性。用于进程控制的原语有:①创建原语;②撤消原语;③挂起原语;④激活原语等。

创建原语(Create Primitive) 父进程或祖先进程借助于创建原语来创建新进程。由于创建一个进程的主要任务是形成进程控制块PCB,所以调用者必须提供PCB中有关参数,以便在创建时填入。这些参数有:进程外部标识符n,初始CPU状态S0,优先数K0,初始内存M0,资源清单R0等。

创建原语的操作过程是,先向PCB集合索取一张空白PCB,并获得PCB的内部名字i。若该进程的程序不在内存中,应将它从后援存储器调入内存,然后把调用者提供的参数以及其父进程的内部名称填入PCB,并登记帐数据,置该进程为"静止就绪"状态,最后把它分别插入进程家族和就绪队列PL中,创建原语可描述为:

```
procedure Create(n,S0,K0,M0,R0,acc)
begin
    i:=Getinternal Name(n);
    Id(i):=n; Priority(i):=K0;
    Cpustate(i):=S0;
    Main store(i):=M0;          Resources(i):=R0;
    Status(i):="readys";        Sdata(i):=RL;
    Parent(i):=*;               Progeny(i):=φ;
    Insert(Progeny(*),i);
    Set According Data;
    Insert(RL,i)
end
```

原语中Sdata(i)是一个与进程状态有关的指针,当进程处于执行状态和就绪状态时,它指向就绪队列;当进程处于阻塞状态时,则Sdata(i)指向引起进程阻塞的资源等待队列。

挂起原语(Suspend Primitive) 当需要把某个进程置于挂起状态时,可调用挂起原语。

挂起原语的操作过程如下:以被挂起进程标识符n为索引,到PCB集合中查找该进程控制块PCB,得到该进程的内部名称i,并把进程的状态赋予S。若S为执行状态,则中断处理机把CPU状态保存于PCB的CPU状态保护区中,停止执行该进程,然后把PCB副本存入内存的指定区域a,以便调用者参考。若进程状态是活

动阻塞，则改为静止阻塞，否则把被挂起的进程置成静止就绪。若S为执行，则调度从活动队列中选取优先数最高的进程投入运行。挂起原语可描述为：

```
Procedure Suspend(n,a)
begin:
    i:=Getinternal Name(n);
    s:=Status(i);
    if S="executing" then Stop(i);
    a:=copy PCB(i);
    Status(i):=if s="blockeda" then "blockeds"
                else "readys";
    if s="executing" then Scheduler
end
```

激活原语(Activate Primitive) 激活原语使处于静止状态的进程变为活动，即把"静止就绪"变为"活动就绪"，把"静止阻塞"变为"活动阻塞"。激活原语能且仅能激活某进程自己的"子孙"。同样，激活原语也可有多种方式，如激活一个具有指定标识符的进程，或者激活某进程及其所有的子进程。

由创建原语可知，新创建的进程是处于"静止就绪"状态，若希望该进程能尽早地获得处理机，则应在创建原语后面紧跟一条激活原语，使之变成活动就绪状态，从而引起调度程序的重新调度。激活具有指定标识符的进程的原语可描述为：

```
Procedure Activate(n);
begin
    i:=Getinternal Name(n)
    Status(i):=if Status(i)="readys" then "readya"
                else"blockeda";
    if status(i)="blockeda" then Scheduler
end
```

撤消原语(Destroy Primitive) 当一个进程在完成其任务后应予以撤消，以便及时释放它所占用的资源。

撤消原语的操作过程是，以调用者所提供的标识符n为索引，到PCB集合中去搜索，获得该进程的内部名称i和状态。若进程正处于执行状态，则应立即中断处理机，把CPU状态保存在CPU状态保护区中，停止执行该进程，并设置要重新调度的标志。由Sdata(i)指示该进程所在队列，将它从该队列中消去，而且凡是属于该进程的"子孙"也一律予以撤消。对于被撤消者所占用的资源，若它们是属于撤消者或其祖先的，都应归还。凡是属于被撤消者自己的，则消去它的资源描述块，最后消去被撤消进程的控制块。若调度标志为真，则重新调度。撤消原语的描述如下：

```
Procedure Destroy(n);
begin
    Sched:=false;
    i:=Getinternal Name(n);
    Kill(i);
    if Sched then Scheduler;
end
Procdeure Kill(i);
begin
    if status(i)="executing" then
```

```
        begin Stop(i); Sched:=true end;
        Remove(Sdata(i),i);
        for all s∈ Progeny(i) do Kill(s);
        for all r∈ (Main Store(i)∪Resources(i)) do
        if owned(r) then insert (Avail (Semaphone(r)),data(r));
        for all R ∈created Resources(i) do
              Remove Resource Descriptor(R);
        Remove Process Control block(i);
    end
```

阻塞原语(Block Primitive)和唤醒原语(Wake up Primitive) 由"执行"到"阻塞"和由"阻塞"到"就绪"，分别是在进程控制原语Block 和 Wake up 原语作用下完成的。

当一进程所期待之事件尚未出现时，该进程调用Block 原语把自己阻塞起来。Block 原语的操作过程如下：由于进程正处于执行状态，故应中断处理机，把CPU 状态保存在PCB 的CPU 状态保护区中，停止执行该进程。然后把"活动阻塞"态赋予该进程，并把它插入该事件的等待队列中，再从活动就绪队列中按一定算法选取一进程投入运行。阻塞原语可描述为：

```
    Procedure Block
    begin
        i:=Getinternal Name(n);
        Stop(i);
        Status(i):="blockeda";
        Insert(WL(r),i);
        Scheduler
    end
```

相应地，唤醒原语描述为：

```
    Procedure Wakeup(n);
    begin
        i:=Getinternal Name(n);
        Remove (WL(r),i);
        Status(i):="readya";
        Insert(RL,i)
    end
```

进程的调度 进程的调度就是按一定的算法，动态地把处理机分配给就绪队列中的某一进程，使之执行。

在多道程序系统中，用户进程数往往多于处理机数，这将使它们相互争夺处理机，同时，系统进程也同样需要处理机，这就需要进程调度来协调。进程调度的具体功能应包括：

(1)记住系统中所有进程的状态、优先数和所用资源情况。如果是动态优先数，必须按一定的算法定期地对它计算。

(2)当CPU 空闲时，按一定算法确定应把处理机分配给哪个进程和分配多少时间。当某进程正在执行时，假如有优先数更高的进程进入就绪队列，就须由调度方式来决定，是继续执行原进程，还是把处理机分配给新到的进程，使之优先执行。

(3)执行把处理机分配给进程的操作，即把正在执行的进程的状态改为"就绪"或"阻塞"，并插入相应的队列中。把应获得处理机的进程从就绪队列移走，并改为"执行"状态。

所谓调度方式，是指当一进程正在处理机上执行时，若有某个更为"重要或紧迫"的进程需要进行处理，

亦即，若有优先数更高的进程进入就绪队列时，如何分配处理机。通常有两种进程调度方式：一种是仍然让正在执行的进程继续执行，直到该进程完成或发生某事件(如提出I/O请求)，而进入"完成"或"阻塞"状态时，才把处理机分配给"更重要"或"更紧迫"的进程，使之执行，这种进程调度方式称为非剥夺方式；另一种方式则是"重要或紧迫"的进程一到，便暂停正在执行的进程，立即把处理机分配给它。此即所谓的剥夺调度方式。

进程调度的核心问题是，使用什么算法把处理机分配给进程。进程调度的算法较多，在设计进程调度算法时应考虑的因素也较多，比如，尽量提高资源利用率，减少处理机的空闲时间，对于用户作业采用较合理的平均响应时间，以及尽可能地增强CPU的处理能力等。此外，还应避免有些作业长期得不到响应的情况发生。事实上，上述要求往往是相互矛盾的，因此在设计进程调度算法时只能采取折衷的办法，即宁可使总的调度性能良好，也不愿为获得某一最佳性能，而以其它性能低劣为代价。

进程调度算法 常用的算法有两种：(1)最高优先数优先法；(2)轮转法。

1. 最高优先数优先法(Highest Priority First)

这种进程调度算法是把处理机分配给具有最高优先数的进程。在这种算法中，首先考虑的问题是如何确定进程的优先数，其方法很多，但概括起来有两种：基于静态特性和基于动态特性。

(1)静态优先数。静态优先数是在系统创建进程时确定的，一经确定之后在整个进程运行期间不再改变，确定静态优先数的有关静态特性如下：

① 进程类型。系统中有系统进程和用户进程两类进程，通常系统进程比用户进程具有较高的优先数，特别是某些系统进程，必须赋于一种特权，当它需要处理机时，应尽快得到满足。

② 作业要求的资源。根据作业要求系统提供的处理机时间、内存的大小和I/O设备的数量，来确定作业的优先数。

③ 外部优先数和作业的到达时间。一种用于多道批量处理系统的方法是，当作业进入时，系统把用户作业卡上所提供的外部优先数，赋予该作业及其所创建的进程。另一种方法是根据作业到达的先后，给先到者以最高优先数，使之先执行，形成先来先服务算法(FCFS)。

(2)动态优先数。在进程运行期间，既可按线性也可按非线性来改变进程的优先数。一种按线性方式改变优先数的规则是，就绪队列各进程的优先数以a的速率增加，正在执行的进程其优先数以b的速率改变，通过选择不同的a及b之值，就可形成各种调度算法。

一种非线性改变优先数的规则是，在进程进入系统后的前一阶段，其优先数不变或随时间线性递减，当该进程的等待时间达到某一给定的最大值时，其优先数又突然跃变到某一最高值，从而使该进程能很快投入执行。

进程调度过程描述如下：

```
Procedure Scheduler
begin
  A:if Length(RL)! = 0 then
    begin
        lockout interrupts;
        remove(RL,P);
        length(RL):=length(RL)-1;
        unlock interrupts;
        Status(P):="executing";
        restore(cpustate(p));
    end
    else goto A
end
```

2. 轮转法(Round Robin)

(1)简单轮转法。系统把所有就绪状态进程按FCFS规则排成一个队列。处理机分配给队列第一个进程，执行一个时间片，系统便将它送至就绪队列的末尾，又把处理机分配给就绪队列中下一进程，再执行同样大小的时间片。

(2)可变时间片轮转法。当每一轮开始时，系统便根据就绪队列中已有的进程数目计算一次时间片q值，$q=T/N_{max}$，(其中，T所要求的响应时间，N_{max}为当前进程个数)然后进行轮转，在此期间所到的进程都暂不进入就绪队列，要待到此次轮转完毕后再一并进入。

(3)多队列轮转法。在双就绪队列轮转法中设置两个就绪队列，一个称为前台队列，它具有较高优先数，另一个称为后台队列，它具有较低的优先数。平时进程调度以固定时间片的形式把处理机分配给前台队列中的进程，以保证它们有更多的机会得到服务。仅当前台队列中的进程已全部完成或因等待I/O操作而无进程可执行时，才把处理机时间分配给后台进程，并按突击方式执行。

时间片 在分时系统中，如果系统中有n个同时性用户，则将处理机分配给每个用户并规定执行一给定时间，该时间称为时间片q，即每个用户在$n*q$秒时间内获得q秒处理时间。

在轮转法中，时间片q的大小对进程调度有很大影响。如果q取得足够大，以致所有进程都能在分给它的时间片内执行完毕，则此时的轮转法已退化为FCFS法，这无论是对短作业，还是I/O操作较多的作业都是不利的。随着q的减小，调度性能可随之得到改善，当q值足够小时，就绪队列中的所有进程都能得到同样的服务，服务时间与等待时间之比趋于一个常数。但当q减小到很小时，由一个进程到另一进程的转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。因此，q值必须确定得比较适中，通常为100ms数量级。

时间片的长短由以下4个因素确定：

(1)系统的响应时间：在进程数目一定时，时间片的长短直接比例于系统对响应时间的要求；

(2)就绪队列进程的数目：当系统要求的响应时间一定时，时间片的大小反比于就绪队列中的进程数；

(3)进程的转换时间：若进程的转换时间为t，时间片为q，为保证系统开销不超过某一水平，应使比值t/q不大于某一数值；

(4)计算机的处理能力：计算机的处理能力直接决定了每道程序的处理时间，或者说决定了所需的时间片数，进而也决定了完成一进程所需要的转换次数和转换时间。显然，计算机的速度愈高，时间片就可愈短。

程序的并发执行(共行执行) 所谓共行执行是指程序段的执行在时间上是重叠的，即使这种重叠只有很小一部分，也称两个程序段是共行执行。

为增加系统的处理能力和提高机器的利用率，在现代计算机中广泛采用共行操作技术，使多种硬件设备能并行操作。

把单个程序分成若干个能共行执行的程序段，可带来如下好处：第一，简化程序设计；第二，在多处理机系统中，可加快程序完成的速度；第三，在多道程序环境下，可有效利用系统的资源。

进程的一个基本特征就是共行特征，引入进程的目的就是为了能使程序共行执行，以提高系统资源的利用率。

程序的并发执行具有3个主要特性：

(1)没有封闭性。在共享公共变量时，其执行结果不可再现。

(2)程序与计算不再一一对应。"程序"是指令的有序集合，是"静态"的概念，而计算是指令序列在处理机上的执行过程，是"动态"概念。在并发执行中，一个共享程序可被多个用户作业调用，从而形成了多个计算。

(3)并发程序在执行期间可以互相制约，从而使各个程序的执行过程不再像单道程序系统中那样顺序连贯执行，而具有执行——暂停——执行的活动规律，各程序活动的工作状态与所处环境有密切关系。

临界区 一次仅允许一个进程使用的资源称为临界资源(Critical Resource)，访问临界资源的那段程序能够从概念上分离出来，而被称为临界区。

虽然系统中可同时有许多进程，它们共享各种设备，但很多设备都属于临界资源，如输入机、打印机、磁带机等。

假设有两个进程A、B共享一台打印机，为不使两个进程的输出结果交织在一起，就要使进程A使用打印机提出申请，系统把资源分配给它，就一直为它独占，进程B等待，直到A进程用完并释放后，系统才把打印机分配给进程B使用。

从概念上讲，系统中诸进程在逻辑上是独立的，它们可按各自独立的速度向前推进。但由于它们共享某些物理资源，而产生临界区问题。为禁止两个进程同时进入临界区，必须有软件方法或一个同步机构来协调它们，该软件算法或同步机构应遵循下述准则：

(1)当有若干进程欲进入它们的临界区时，应在有限时间内使其中一进程进入临界区。

(2)每次至多有一个进程处于临界区。

(3)进程仅在临界区内逗留有限的时间。

根据准则(2)和(3)，得到临界区调度原则应是：

(1)当无进程处于临界区内时，允许一进程立即进入临界区。

(2)当有一进程处于临界区内时，其它试图进入临界区的进程必须等待。

(3)当有一进程离开临界区时，若有等待进程，则允许其中一个进入临界区。

由此可知，共行进程使用临界区时，必须互斥，即保证每次至多有一个进程进入临界区。

进程同步 同步是进程间共同完成一项任务时直接发生相互作用的关系，也就是说，这些具有伙伴关系的进程在执行的时间次序上必须遵循一定的规律。所谓进程同步是指，相互合作的进程之间须要交换一定的信息，当某进程未获得其合作进程所发来的信息之前，该进程等待，直到该信息到来才被唤醒继续执行，从而保证了诸进程的协调运行。

假定有两个合作进程共同使用了单缓冲，计算进程对数据进行计算，打印进程打印计算结果。当计算进程对数据的计算尚未完成，未把结果送入缓冲区之前，打印进程无法执行打印操作。一旦计算进程把计算结果送入缓冲区时，就应给打印进程发出一信号，打印进程收到该信号后，便可从缓冲区取出计算结果打印，反之，在打印进程尚未把缓冲区的计算结果取出打印之前，计算进程也不能再把下一次计算结果送入缓冲区。这同样需要在打印进程取走缓冲区中的计算结果时，给计算进程发送一个信号，计算进程收到该信号后才能将下一次计算结果送入缓冲区。计算进程和打印进程之间便是用这种信号量方式实现同步的。

设置和测试(Test and Set) 在测试与设置方法中，大多数同步机构都是采用一物理实体，如锁或信号量来代表某种临界资源的状态。变量W称为"锁"或"锁位"，W=0表示资源可用，W=1则表示资源已被使用。进程使用临界资源的操作应按如下三步进行：

(1)测试W之值，若W=1，表示资源已被占用，返回重新测试；若W=0，表示资源可用，则置位W(关锁)；

(2)进入临界区，访问临界区资源；

(3)退出临界区，复位W(开锁)。

一个进程在进入临界区前应先进行关锁操作，在退出临界区时应执行开锁操作。其中的关锁和开锁原语可描述为：

关锁原语 lock w：

```
L: if w=1 then goto L else w:=1
```

开锁原语 unlock w：

```
w:=0;
```

下面给出一个两共行循环进程用关锁和开锁原语来实现进程互斥的例子。

```
begin integer w;
  w:=0;
```

```
cobegin
begin
L1:lock w;
  CS1;
  unlock w;
  remainder of process 1;
  go to L1
end
begin
L2:lock w;
  CS2;
  unlock w;
  remainder of process 2;
  go to L2
end
coend
end
```

虽然上述的同步机构能有效地保证进程互斥，但对于不能进入临界区的进程，却在不断地测试锁位W，以等待它为0，这就造成了处理机的浪费。一种改进的方法是，每当进程n所需的临界资源被占用时应使进程阻塞，并把它插入到W的等待队列WL中，然后把处理机分配给其它进程执行。一旦该资源被释放，便检查W等待队列WL空否，若空，则复位W以表示该资源已可用；若不空，则从WL队列中移出一个进程q，将它置成活动就绪状态后插入就绪队列RL。做了上述修改后的关锁和开锁原语描述为：

```
lock w:if w=1 then
    begin
        Block n;Insert(WL,n);Scheduler
    end
    else w:=1;
unlock w:if wL≠Ω  then
      begin
        Remove(wL,q);
        Status(q):="readya";
        Insert(RL,q)
      end
      w:=0;
```

信号量(Semaphores)与PV操作 在操作系统中信号量是表示资源的物理实体，是一个与队列有关的整型变量，其值仅能由P和V操作来改变。操作系统利用它的状态对进程和资源进行管理。根据用途之不同可把信号量分为：

(1)互斥信号量：它联系着一组共行进程，其初始值为1。每个进程均可对之施加P和V操作，通常，它是为实现进程互斥而设置。

(2)资源信号量：它也联系着一组共行进程，但其初始值为0，或为某个正整数n，用以表示资源的数目，通常用于进程同步。

信号量的数值仅能由P，V操作加以改变。每执行一次P操作，信号量的数值S减1，此时若S≥0，则进程

q继续执行；若S<0，则阻塞该进程，并把它插入该信号量的等待队列Q中，重新进行调度。P操作可描述为：

```
Procedure P(S)
begin
    Lock out interrupts;
    S:=S-1;
    if S < 0 then
    begin
      Status(q):=Blockeda;
      Insert(Q,q);
      Unlock interrupts;
      Scheduler
    end
    else Unlock interrupts
end
```

每次对该信号量执行一次V操作后，S的内容加1，若加1后S>0，则q进程继续执行。若S≤0，则从信号量等待队列Q中移出一个进程R，把活动就绪状态赋予该进程，或者说将该进程唤醒，然后把它插入就绪队列RL中，q进程继续执行。V操作可描述为：

```
Procedure V(S)
begin
  Lock out interrupts;
  S:=S+1;
  if S<=0 then
  begin
    Remove(Q,R);
    Status(R):=Readya;
    Insert(RL,R);
    Length(RL):=Length(RL)+1
  end
  Unlock interrupts
end
```

应当注意，P和V操作是同步原语，在执行期间不可分割。

从物理概念上说，S>0时的数值表示某类可用资源的数量。执行一次P操作意味着请求分配一个单位的资源，因此描述为S:=S-1。当S<0时表示已无资源，因此，请求该资源的进程将被阻塞，把它排在信号量S的等待队列Q中，此时S的绝对值等于信号量队列Q上的进程数目。而执行一次V操作意味着释放一个单位资源，故作S:=S+1。若S<=0表示信号量请求队列Q中仍有因请求该资源而被阻塞的进程，因此就应把队列Q中的第一个进程唤醒使之转至就绪队列。

生产消费者问题(Producer - Consumer Problems) 生产消费者问题是很多共行进程内在关系的一种抽象。例如，生产者可以是计算进程，消费者是打印进程。在输入时可以认为输入进程是生产者，计算进程是消费者。因此，生产者-消费者问题是有很大实用价值的，可以通过一个有界缓冲区把一群生产者P1，P2，…，Pn和一群消费者C1，C2，…，Cm联系起来，如图8.1-3所示。

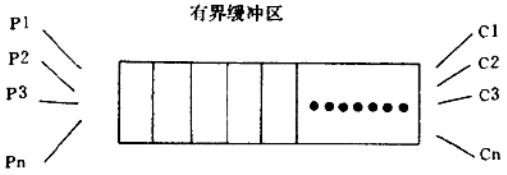

图8.1-3 生产消费者问题

假定这些生产者和消费者是互相等效的，只要缓冲区未满，生产者在执行P(avail)操作之后avail≥0，就可把产品送入缓冲区；类似地，只要缓冲区未空，消费者在执行了P(full)操作后full≥0，便可从缓冲区取走物品并消耗它。仅当缓冲区满时，生产者由于在执行P(avail)操作后avail<0而被阻塞；类似地，缓冲区空时消费者进程执行P(full)操作后因full<0而被阻塞。为了实现上述两进程互斥地进入临界区，设置两个私用信号量和一个公用信号量如下：

(1)公用信号量mutex：赋予其初值为1，表示没有进程进入临界区，它用于实现进程互斥；

(2)私用信号量full：赋予其初值为0，用以计算产品数目；

(3)私用信号量avail：赋予其初值为n，表示可利用的缓冲区数目。

生产者—消费者问题的描述如下：

```
begin integer mutex,avail,full;
  mutex:=1;avail:=n;full:=0;
  cobegin
    producer:begin
      L1:produce next product;
        P(avail);
        P(mutex);
        add to buffer;
        V(full);
        V(mutex);
        go to L1
      end
    consumer:begin
      L2:P(full);
        P(mutex);
        take from buffer;
        V(avail);
      V(mutex);
    consume product;
      go to L2
      end
  coend
end
```

应当注意，无论在生产者进程，还是消费者进程中，V操作的次序都无关紧要，但P操作的却不能颠倒，否则将可能造成死锁。

此外，在上述描述中，full和avail都是一般信号量，但亦可将其中一个改为二元信号量。