第 1 章　操作系统引论

现代计算机系统由一个或多个处理机、主存、磁盘、打印机、键盘、鼠标、显示器、网络接口以及各种其他输入/输出设备组成，是一个复杂的系统。为了管理计算机系统，使计算机的使用更加方便，人们给计算机安装了软件，称为操作系统。

1.1　操作系统的定义和作用

操作系统利用一个或多个处理机的硬件资源，为系统用户提供一组服务，它还代替用户来管理辅助存储器和输入/输出(input/output，I/O)设备。

1.1.1　操作系统的定义

计算机软件分为系统软件和应用软件两大类。系统软件用于管理计算机本身和应用程序，应用软件是为满足用户特定需求而设计的软件。操作系统和系统工具软件构成了系统软件。

操作系统在计算机中的位置如图 1-1 所示。它运行在裸机之上，允许用户运行其他程序，诸如 Web 浏览器、电子书阅读器和视频播放器等。

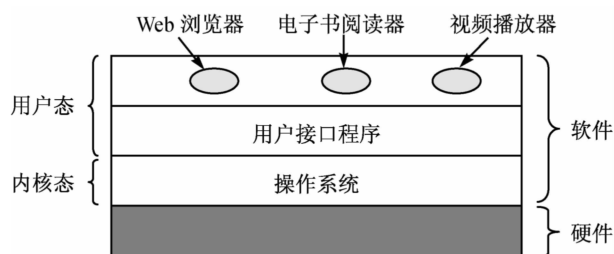

图 1-1　操作系统在计算机中的位置

操作系统是一种运行在内核态的软件。操作系统执行两个基本独立的任务，即为应用程序提供一个资源集的清晰抽象，并管理这些硬件资源。无论从哪个角度看待操作系统，它都完成 3 个目标：

- 方便：操作系统使计算机易于使用。
- 有效：操作系统允许以更有效的方式使用计算机系统资源，包括硬件和软件资源。
- 扩展的能力：在构造操作系统时，应该允许在不妨碍服务的前提下有效地开发、测试和引进新的系统功能。

下面从两方面介绍操作系统的这 3 个目标，以便更好地理解操作系统。

(1)作为用户与计算机接口的操作系统。为用户提供各种应用的硬件和软件可以看做

是一种层次结构，如图 1-2 所示。应用程序的用户（最终用户），通常并不关心计算机的硬件细节。因此，最终用户把计算机系统看做是一组应用程序。一个应用程序可以用一种程序设计语言描述，它是由程序员开发的。如果需要用一组完全负责硬件的机器指令开发应用程序，将会是一件非常复杂和困难的任务。为简化这个任务，需要提供一些系统程序，其中一部分称为实用工具，它们实现了在创建程序、管理文件和控制 I/O 设备中经常使用的功能。程序员在开发应用程序时将使用这些软件；应用程序在运行时，将调用这些实用工具以实现特定的功能。最重要的系统程序是操作系统，操作系统为程序员屏蔽了硬件细节，并为程序员使用系统提供了方便的接口。它可以作为中介，使程序员和应用程序更容易地访问和使用这些功能与服务。

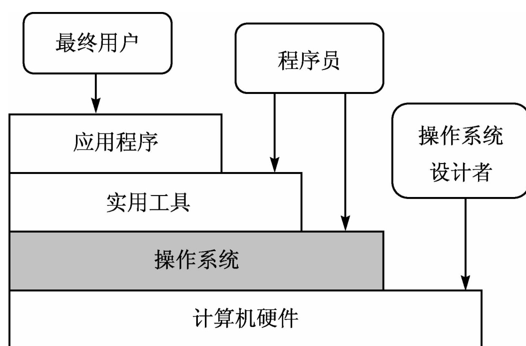

图 1-2　计算机系统的层次结构

（2）作为资源管理者的操作系统。把操作系统看做是向应用程序提供基本接口，这是一种自顶向下的观点。按照另一种自底向上的观点，操作系统则用来管理一个复杂系统的各个部分。现代操作系统包含处理机、存储器、时钟、磁盘、鼠标、网络接口、打印机以及许多其他设备。从这个角度看，操作系统的任务是在相互竞争的程序之间有序地控制对处理机、存储器以及其他 I/O 接口设备的分配。

当一个计算机有多个用户时，管理和保护存储器、I/O 设备以及其他资源的需求变得强烈起来，因为用户间可能会互相干扰。另外，用户通常不仅要共享硬件，还要共享信息（文件、数据库等）。简而言之，操作系统的这一种观点认为，操作系统的主要任务是记录哪个程序在使用什么资源，对资源请求进行分配，评估使用代价，并且为不同的程序和用户调解互相冲突的资源请求。

综上所述，操作系统是计算机系统中的系统软件，管理和控制计算机系统中的硬件和软件资源，合理地组织计算机的工作流程，以便有效利用这些资源为用户提供一个功能强、使用方便的工作环境，从而在计算机与用户之间起到接口的作用。

1.1.2　操作系统的作用

操作系统是管理计算机硬件与软件资源的程序，同时也是计算机系统的内核。操作系统是一个庞大的管理控制程序，管理着系统中的各种资源并为用户提供接口来操作计算机。

1. 管理系统中的各种资源

在计算机系统中，所有硬件部件（如 CPU、存储器、I/O 设备等）称为硬件资源，而程序和数据等信息称为软件资源。操作系统对每一种资源的管理都必须进行以下几项工作：

(1)监视资源。监视资源包括需要知道该资源有多少,资源的状态如何,它们都在哪里,谁在使用,可供分配的又有多少,资源的使用历史,等等。

(2)决定分配资源策略。决定分配资源策略包括选择哪种资源分配策略,决定谁有权限可以获得这种资源,何时可以获得,可以获得多少,如何退回资源,等等。

(3)分配资源。分配资源是指按照已决定的资源分配策略,对符合条件的申请者分配某种资源,并进行相应的管理事务处理。

(4)回收资源。回收资源是指在使用者放弃某种资源之后,对该种资源进行善后处理,如果是可重复使用的资源,则进行回收、整理,以备再次使用。

2. 为用户提供友好的界面

操作系统必须为最终用户和系统用户的各种工作提供友好的界面,以方便用户的工作。典型的操作系统界面有两类:

(1)命令行界面。如 UNIX 和 MS-DOS 操作系统,在这种操作系统环境下,用户通过键盘输入完成某种工作的命令,操作系统分析命令的正确性,然后执行命令并返回执行结果。

(2)图形化界面。如 Windows 操作系统,在其提供的图形化界面环境下,用户可通过单击某个按钮或选择某个菜单来完成相应的命令。

1.2 操作系统的发展过程

操作系统与计算机系统的发展息息相关。操作系统从最早的批处理阶段开始,经历了分时机制、多处理机等阶段,后来又添加了多处理机协调功能,甚至是分布式系统的协调功能。

1.2.1 无操作系统的计算机系统

第一台真正的数字计算机是英国数学家 Charles Babbage(1791—1871 年)设计的。尽管 Babbage 花费了几乎一生的时间和财产,试图建造他的“分析机”,但是他始终未能让机器正常的运转,因为它是一台纯数字计算机,他所在时代的技术不能生产出他所需要的高精度机械设备。毫无疑问,这台“分析机”没有操作系统。

从 Babbage 失败之后一直到第二次世界大战期间,数字计算机的建造几乎没有什么进展,这次战争引发了有关计算机研究的爆炸性展开。衣阿华州立大学的 John Atanasoff 教授和他的学生 Clifford Berry 建造了第一台可工作的数字计算机,该机器使用了大约 300 个真空管。同时,其他一些大学院校也建造了各自的数字计算机。这些机器非常庞大,往往使用数万个真空管,占据几个房间,但其运算速度却不如现在最便宜的个人计算机。

在计算机出现的早期,每台机器都有一个小组专门来设计、制造、编程、操作和维护。编程全部采用机器语言,通过在一些插板上的硬连线来控制其基本的功能,这时没有编程语言,操作系统更是闻所未闻。

到 20 世纪 50 年代早期,出现了穿孔卡片,这时就可以不用插板,而是将程序写在卡片上,然后读入计算机。

1.2.2 单道批处理系统

20 世纪 50 年代发明的晶体管极大地改变了计算机的状况。这时的计算机已经很可靠，厂商可以成批地生产计算机并卖给客户，客户可以长时间地使用它来完成一些有用的工作。至此，第一次将设计人员、生产人员、操作员、程序员和维护人员分开。

这个时期，计算机被安置在空调房间里，并配有专人操作。由于其价格昂贵，仅有少数大公司、主要的政府部门和大学才买得起。运行一个作业（一个或一组程序）时，程序员首先将程序写在纸上（用 FORTRAN 或汇编语言），然后用穿孔机制成卡片，最后将这些卡片交给操作员。

操作员从卡片上读入任务，计算机运行完成当前任务后，其计算结果从打印机上输出，操作员从打印机上取得运算结果并送到输出室，程序员就可以从输出室得到运算结果，然后，操作员再从卡片上读入另一个任务。当操作员在机房里走来走去时，许多机时被浪费掉了。

由于当时计算机非常昂贵，很自然地，人们开始想办法减少机时的浪费，批处理系统由此产生。其思想是：在作业输入室收集到较多的作业后，使用一台相对廉价的计算机将它们读到磁带上，另外用较昂贵的计算机来完成真正的计算。

在收集到一批作业之后，输入磁带被送到机房里装到磁带机上。操作员随后装入一个特殊的程序（现代操作系统的前身），它从磁带上将第一个作业读入并运行，其输出写到第一盘输出磁带上，而不是打印出来。每个作业结束后，此特殊的程序自动地读入下一个作业运行。当这一批作业完全结束后，操作员取下输入和输出磁带，将输入磁带换成写有下一批作业的输入磁带，然后把输出磁带拿到一台机器上进行脱机打印。

此时的批处理系统称为单道批处理系统，又称为监督程序或管理程序，用于管理应用程序的运行。

1.2.3 多道批处理系统

由于在单道批处理系统中，一个作业单独进入内存并独占系统资源，直到运行结束后下一个作业才能进入内存，当作业进行 I/O 操作时，CPU 只能处于等待状态，因此，CPU 利用率较低，尤其是对于 I/O 操作时间较长的作业。为了提高 CPU 的利用率，在单道批处理系统的基础上引入了多道程序设计（multiprogramming）技术，这就形成了多道批处理系统，即在内存中可同时存在若干道作业，作业执行的次序与进入内存的次序无严格的对应关系，因为这些作业是通过一定的作业调度算法来使用 CPU 的，一个作业在等待 I/O 处理时，CPU 调度另外一个作业运行，因此 CPU 的利用率得到了显著地提高。

多道批处理系统的优点是由于系统资源为多个作业所共享，其工作方式是作业之间自动调度执行，并在运行过程中用户不干预自己的作业，从而大大提高了系统资源的利用率和作业吞吐量。其缺点是无交互性，用户一旦提交作业就失去了对其运行的控制能力，而且是批处理的，作业周转时间长，用户使用不方便。

1.2.4 分时系统

为了克服多道批处理系统的不足，引入了分时操作系统。分时处理，又称会话型处理，是在多道程序设计基础上发展起来的一种处理方式。它把时间分隔技术应用到 CPU 的调度上，形成了一种新的操作系统。第一个真正的分时处理系统是美国麻省理工学院研制的

CTSS(compatible time sharing system)，它支持位于不同终端的多个用户同时使用一台计算机，彼此独立互不干扰，使用户感到好像整台计算机全为他所用。

1.2.5　实时系统

分时系统为交互式作业提供了快速的响应服务，但还不能满足某些对响应时间要求非常严格的任务需要。例如，炼钢/炼油控制系统、航空网络售票系统等，对任务的响应时间有更严格的要求。

为了满足对响应时间要求严格的任务的需求，人们研发出了实时系统。实时操作系统指使计算机能及时响应外部事件的请求，在严格规定的时间内完成对该事件的处理，并控制所有实时设备和实时任务协调一致地工作的操作系统。

实时操作系统要追求的目标是对外部请求在严格时间范围内作出反应，有高可靠性和完整性。其主要特点是资源的分配和调度首先要考虑实时性，然后才是效率。此外，实时操作系统应有较强的容错能力。

1.3　操作系统的类型

目前计算机上常见的操作系统有 OS/2、UNIX、XENIX、Linux、Windows、Netware 等。操作系统大致可分为 8 种类型，即批处理操作系统、分时操作系统、实时操作系统、微机操作系统、多处理机操作系统、网络操作系统、分布式操作系统以及嵌入式操作系统。

1.3.1　批处理操作系统

批处理操作系统是创建于 20 世纪 50 年代末期的第一代操作系统，主要是受到了早期系统效率低的启发。由于早期系统在装入、汇编和执行程序的每一步都需要操作人员的手工辅助，这就消耗了大量时间，从而导致了极其昂贵的硬件设备的低效使用。

解决方法是使用一个中央控制程序对标准的装入、汇编、执行的过程进行自动化。这个控制程序可以发现和装入所需的系统程序——汇编器、编译器、链接器或例程库等，并能处理作业到作业的自动切换。这样可以递交多个作业由系统同时处理，典型方式是使用一批打孔卡片。这个控制程序被称为批处理操作系统。

新的硬件技术极大地促进了这些系统的发展。真空管被晶体管所取代，使得机器体积显著减小，并更可靠、更便宜。I/O 处理也得到了明显的改进。最重要的一个硬件创新是 I/O 处理机或 I/O 通道，它类似于 CPU 的处理机，但它是一种带有专用 I/O 处理的较小的指令集合。I/O 处理机把主处理机(CPU)从频繁的低层交互(与 I/O 设备的交互)中解脱出来。当收到来自 CPU 的高层 I/O 请求时，I/O 处理机有效地提高了 CPU 和许多 I/O 设备的并行操作，进而通过并行化改善了系统的整体性能。

起初，CPU 周期性地询问 I/O 处理机的状态，看它是处于忙碌态还是发生了某种错误。但很快人们就发现，如果采用一种机制，仅当 I/O 处理机需要 CPU 时(或者报告问题或者接收新的命令)才通知 CPU，系统可以更加高效地运转。解决方法是引入了中断，这是一种硬件信号，由 I/O 处理机发送给 CPU 以得到 CPU 的立即注意。后来把中断扩展为允许 CPU 可以快速响应多种不同的情况，例如，除数为零、无效的操作符、违反内存保护等。

为进一步流线化批处理系统的操作,人们编写编译器以产生可重定位的而不是绝对的代码。后来编写了链接器,它能组合之前单独编译后的程序(包括例程库),而无须重新编译。通过以上方法,仅当管理 I/O 批量作业、设置非标准任务和系统失败需要采取修正行为时,系统才需要用户的维护。程序员逐渐被禁止进入机房,那里成为计算机管理员(一个新的专家)的独占领域。

术语“批”至今仍在广泛使用,但其含义已经发展为表示非交互计算。一个批处理作业是指它不需要任何用户交互,把它提交给系统,意味着系统会选择在一个最合适的时间执行它。例如,系统一般会在一个比交互式作业低的优先级下执行一个批处理作业。系统也可以选择推迟它的执行,直到系统具有足够的空间/时间或直到它已经累积了一批合适的执行作业。另外,用户可以通过高作业级别使得作业可以在以后某个时间(如晚上)开始。

1.3.2 分时操作系统

分时操作系统的工作方式是:一台主机连接了若干个终端,每个终端有一个用户在使用。用户交互式地向系统提出命令请求,系统接受每个用户的命令,采用时间片轮转方式处理服务请求,并通过交互方式在终端上向用户显示结果。用户根据上步结果发出下步命令。分时操作系统将 CPU 的时间划分成若干个片段,称为时间片。操作系统以时间片为单位,轮流为每个终端用户服务。每个用户轮流使用一个时间片而使每个用户并不感到有别的用户存在。分时系统具有多路性、交互性、独占性和及时性的特征。多路性是指同时有多个用户使用一台计算机,宏观上看是多个人同时使用一个 CPU,微观上是多个人在不同时刻轮流使用 CPU;交互性是指用户根据系统响应结果进一步提出新请求(用户直接干预每一步);独占性是指用户感觉不到计算机为其他人服务,就像整个系统为他所独占;及时性是指系统对用户提出的请求及时响应,它支持位于不同终端的多个用户同时使用一台计算机,彼此独立互不干扰。

常见的通用操作系统是分时系统与批处理系统的结合。其原则是:分时优先,批处理在后。“前台”响应需频繁交互的作业,如终端的要求;“后台”处理时间性要求不强的作业。

1.3.3 实时操作系统

实时操作系统的特征是将时间作为关键参数。例如,在工业过程控制系统中,工厂中实时计算机必须收集生产过程的实时数据并用有关数据控制机器;汽车在装配线上移动时,必须在限定的时间内进行规定的操作,如果焊接机器人焊接得太早或太迟,都会毁坏汽车。如果某个运行必须绝对地在规定的时刻(或规定的时间范围)发生,这是硬实时系统。可以在工业过程控制、民用航空、军事以及类似应用中看到很多这样的系统。这些系统必须提供绝对保证,让某个特定的动作在给定的时间内完成。

除了硬实时系统还有软实时系统,在这种系统中,偶尔违反截止时间是不希望的,但可以接受,并且不会引起任何永久性的损害。数字音频或多媒体系统就是这类系统。数字电话也是软实时系统。

1.3.4 微机操作系统

随着超大规模集成电路的发展产生了微机,安装在微机上的操作系统称为微机操作系统。最早出现的微机操作系统,是在 8 位微机上的 CP/M。后来出现了 16 位微机,相应地也就出现了 16 位微机操作系统。当微机发展到 32 位时,又出现了 32 位的微机操作系统。

可见微机操作系统可按微机的字长分成 8 位、16 位和 32 位的微机操作系统。但也可以把微机操作系统分为单用户单任务操作系统、单用户多任务操作系统和多用户多任务操作系统。

1.3.5 多处理机操作系统

为了获取更强的计算能力，一种很重要的方式是将多个 CPU 连接成单个的系统。依据多 CPU 连接和共享方式的不同，这些系统称为并行计算机、多计算机或多处理机。它们需要专门的操作系统，不过通常采用的操作系统是配有通信、连接和一致性等专门功能的服务器操作系统的变体。

个人计算机中近年来出现了多核芯片，所以常规的台式机和笔记本电脑操作系统也开始与小规模的多处理机打交道，而核的数量正在与时俱进。由于先前多年的研究，已经产生了不少关于多处理机操作系统的知识，将这些知识运用到多核处理机系统中不存在困难，困难在于要有能够运用所有这些计算能力的应用。许多主流操作系统，包括 Windows 和 Linux，都可以运行在多核处理机上。

1.3.6 网络操作系统

网络操作系统是基于计算机网络的，是在各种计算机操作系统上按网络体系结构协议标准开发的软件，包括网络管理、通信、安全、资源共享和各种网络应用。其目标是相互通信及资源共享。在网络操作系统支持下，网络中的各台计算机能互相通信和共享资源。其主要特点是与网络的硬件相结合来完成网络的通信任务。

1.3.7 分布式操作系统

大量的计算机通过网络被连接在一起，可以获得极高的运算能力及广泛的数据共享。这种系统被称做分布式操作系统。它在资源管理、通信控制和操作系统的结构等方面都与其他操作系统有较大的区别。由于分布式计算机系统的资源分布于系统的不同计算机上，操作系统对用户的资源需求不能像一般的操作系统那样等待有资源时直接分配而是要在系统的各台计算机上搜索，找到所需资源后才可进行分配。对于有些资源，如具有多个副本的文件，还必须考虑一致性。所谓一致性是指若干个用户对同一个文件同时读出的数据是一致的。为了保证一致性，操作系统必须控制文件的读/写操作，使得多个用户可同时读一个文件，而任一时刻最多只能有一个用户在修改文件。分布式操作系统的通信功能类似于网络操作系统。由于分布式操作系统不像网络操作系统分布得很广，同时分布式操作系统还要支持并行处理，因此它提供的通信机制和网络操作系统提供的有所不同，它要求通信速度高。分布式操作系统的结构也不同于其他操作系统，它分布于系统的各台计算机上，能并行地处理用户的各种需求，有较强的容错能力。

1.3.8 嵌入式操作系统

嵌入式系统用来控制设备在计算机中的运行，这种设备不是一般意义上的计算机，并且不允许用户安装软件。典型的例子有微波炉、电视机、汽车、DVD 刻录机、移动电话以及 MP3 播放器一类的设备。区别嵌入式系统与掌上设备的主要特征是不可信的软件不能在嵌入式系统上运行。用户不能给自己的微波炉下载新的应用程序——所有的软件都保存在

ROM 中,这意味着应用程序之间不存在保护,这样系统就获得到了某种简化。在这个领域中,主要的嵌入式操作系统有 QNX 和 VxWorks 等。

1.4 操作系统的基本特征

操作系统的基本特征包括并发性、共享性、虚拟性和异步性。

1.4.1 并发性

并发性,也称共行性,是衡量操作系统处理多个同时性活动的一个重要指标。两个或多个事件在同一时刻发生称为并发。并发性是指在一段时间内,有多个程序同时进行,但是在单处理机系统中,每一时刻只能有一个程序在执行,所以这些程序只能分时执行;如果计算机系统中有两个或多于两个的处理机,这些程序会被分配到多个处理机同时执行,以实现多任务同时执行。例如,计算机可能同时在进行如下操作:打印文件,复制文件,下载文件。

并发同时也会产生一些问题:如何在多个任务的转换中,保证其互相不受影响以及相互间的制约和同步,为了使并发活动能有条不紊地进行,操作系统要实现多任务间的管理和控制。

1.4.2 共享性

共享指多个任务对资源的同时使用,通常,并发活动会要求共享资源。在应对并发活动时,向每个用户提供一份资源副本并不总是合理的,且多个副本会存在同步困难的问题,因此,操作系统要实现资源的共享。

共享通常有以下两种方式:

(1)同时访问共享。在一些系统资源中,允许在同一段时间内,有多个程序同时访问数据。这时的同时,指的仍然是宏观上的,微观上,这些程序仍然是交替对资源进行访问,由操作系统把任务分给处理机。

(2)互斥访问共享。系统中的有些资源是无法共享的,如打印机。它们的共享势必会造成任务的混乱,所以需要规定它们在同一段时间内只能由一个程序进行访问。

并发与共享密不可分,它们互相依存。并发是为了资源的共享,而共享实现的条件和底层机制基础又是并发。如果操作系统没有很好地处理并发和共享,势必会影响程序的正常运行,甚至无法运行。

1.4.3 虚拟性

在操作系统中,内存、CPU 和外设都采用了虚拟技术,虚拟技术在逻辑上扩充了物理设备的数量。虚拟将一个实体映射为多个逻辑实体,前者客观存在,而后者实际上并不存在。并发性就是虚拟的一个应用:在多道程序设计中,虽然处理机事实上只能同时处理一个任务,但是通过并发技术,好像同时有多个处理机在进行工作,这就是利用虚拟技术将一个 CPU 映射为多个逻辑上的 CPU。同理,还有虚拟存储器等。

1.4.4 异步性

异步性也称为随机性,在多道程序设计中,由于并发性的实现机制,任务通常并不是“一

气呵成”的，而是时而中断，时而运行。例如，一个进程在运行中，需要访问指定资源或者某个事件的发生才能进行下一步操作。在等待的过程中，它会被操作系统暂停，将 CPU 抽取出去执行其他的进程。于是，有新的问题留给了操作系统，程序什么时候开始执行，什么时候暂停，以怎样的速度执行，总共要执行多长时间，这些需要操作系统能合理的处置。

并发性给操作系统带来了潜在的危险，它们在异步的过程中，会产生时间相关的错误，因此确保能捕捉到任何一种事件序列是操作系统的一项重要任务。

1.5　操作系统的主要功能

操作系统的主要功能是资源管理、程序控制和人机交互等。计算机系统的资源可分为设备资源和信息资源两大类。设备资源指的是组成计算机的硬件设备，如中央处理机、主存储器、磁盘存储器、打印机、磁带存储器、显示器、键盘和鼠标等。信息资源指的是存放于计算机内的各种数据，如文件、程序库、知识库、系统软件和应用软件等。

1.5.1　处理机管理功能

处理中断事件是处理机管理的第一项工作，硬件只能发现中断、捕捉中断，并产生中断信号，但却无法对其进行处理，因此需要操作系统来处理硬件发生的中断信号。

捕捉到中断信号之后，处理机要调度任务。在单任务下，处理机仅需要支持一个用户一个任务，工作内容十分简单，但是在多用户多任务下情况就完全不同了，操作系统需要组织多个作业，需要解决处理机的调度、分配和回收。近年来，各种各样的多处理机系统使得处理机的管理更加复杂，为实现对处理机的管理，操作系统引入了进程的概念。

处理机的分配和执行都是以进程为基本单位的，随着并行技术的发展，操作系统进一步引入了线程的概念。因此对处理机的管理可以归纳为对进程和线程的管理，具体包括：进程的控制和管理，进程同步和互斥，进程通信，进程死锁，处理机调度，线程控制和管理等。

由于操作系统对处理机的管理采用了不同的策略，其所能提供的作业方式也不同，可分为批处理方式、分时处理方式、实时处理方式等，因此，派生出各种各样的操作系统。

1.5.2　存储器管理功能

管理存储器资源是存储器管理的主要任务，它为多道程序运行提供有力的支撑。存储器管理的主要功能包括：

(1)存储分配。存储管理根据用户程序的需要给它分配存储器资源。

(2)存储共享。存储管理能让主存中的多个用户程序实现存储资源的共享，以提高存储器的利用率。

(3)存储保护。存储管理要把各个用户程序相互隔离起来互不干扰，更不允许用户程序访问操作系统的程序和数据，从而保护用户程序存放在存储器中的信息不被破坏。

(4)存储扩充。由于物理内存容量有限，难以满足用户程序的需求，存储管理还应该能从逻辑上来扩充内存储器，为用户提供一个比内存实际容量大得多的编程空间，方便用户的编程和使用。

操作系统的这一部分功能与硬件存储器的组织结构和支撑设施密切相关，操作系统设

计者应根据硬件情况和用户使用需要,采用相应的有效存储资源分配策略和保护措施。

1.5.3 设备管理功能

设备管理的主要任务是管理各类外围设备,完成用户提出的各种 I/O 请求,加快 I/O 信息的传送速度,发挥 I/O 设备的并行性以提高 I/O 设备的利用率以及提供每种设备的设备驱动程序和中断处理程序,向用户屏蔽硬件使用细节。为实现这些任务,设备管理应该具有以下功能:

(1)提供外围设备的控制与处理。

(2)提供缓冲区的管理。

(3)提供外围设备的分配。

(4)提供共享型外围设备的驱动。

(5)实现虚拟设备。

1.5.4 文件管理功能

文件管理是对系统的信息资源的管理。在操作系统中,通常把程序和数据以文件形式存储在外存储器上,供用户使用。这样,外存储器上保存了大量文件,对这些文件如不能进行良好的管理,就会导致文件混乱或被破坏,造成严重后果。为此,在操作系统中提供了文件管理功能,它的主要任务是对用户文件和系统文件进行有效管理,实现按名存取;实现文件的共享、保护和保密,保证文件的安全性;提供给用户一套能方便使用文件的操作和命令。具体来说,文件管理要完成以下任务:

(1)提供文件逻辑组织机制。

(2)提供文件物理组织机制。

(3)提供文件的存取方法机制。

(4)提供文件的使用方法机制。

(5)实现文件的目录管理。

(6)实现文件的存取控制。

(7)实现文件的存储空间管理。

1.5.5 用户接口

为了提供计算机与用户之间的交互,操作系统提供各种接口,通常包括程序接口、命令接口和图形接口。

现在的操作系统大多提供图形接口,如 Windows、Mac、Linux 等。

1.6 当前主流操作系统简介

在计算机发展史上,出现过许多不同的操作系统,其中最为常用的有 3 种:Windows、UNIX、Linux。

1.6.1 Windows 操作系统

Windows 是由 Microsoft 公司在 20 世纪 80 年代末期开发的世界上最流行的图形界面

操作系统。Windows操作系统采用事件驱动工作方式,使用成熟的图形界面作为用户的工作桌面,把应用程序的名称和具有的功能用图标形象地标注在桌面上。用户通过鼠标操作就可以运行应用程序。Windows操作系统通过使用CPU时间轮转分时技术、虚拟内存技术和抢先式多任务运行技术来实现多任务功能。Windows系统还拥有较完整的网络功能,使之成为一个桌面网络操作系统,进行局域网络的建设。Windows包括Windows 3.2、Windows 95、Windows 98、Windows me、Windows 2000、Windows XP、Windows 2003以及较新版的Windows Vista和Windows 7。

1.6.2　UNIX操作系统

UNIX于20世纪60年代末期在AT&T贝尔实验室问世,UNIX是一个可以在个人计算机、小型计算机、中型计算机、巨型计算机和工作站上安装的操作系统。UNIX操作系统的问世和发展对计算机硬件和软件的发展作出了巨大的贡献并产生了深刻的影响。

它具有4个显著的特点:

(1)UNIX是可移植的操作系统。它可以不经过较大的改动而方便地从一个平台移植到另一个平台。在UNIX操作系统下开发的应用程序移植也同样方便,其根本原因是UNIX主要部分是用C语言编写的(而不是用于特定操作系统的机器语言),采用C语言编写操作系统使得操作系统的兼容性非常好。

(2)UNIX拥有一套功能强大的工具(命令)集。它们能够组合起来去解决许多问题,而这一工作在其他操作系统中则需要通过编程来完成。

(3)UNIX对设备的操作与具体设备无关。用户不必关心设备和特性,因为操作系统本身就包含了设备的驱动程序,这意味着它可以方便地配置和运行任何设备。

(4)操作系统是开放式的。任何人只要遵守系统的标准都可以给操作系统编写驱动程序和系统代码,并且UNIX的源代码是公开的。

概括地说,UNIX具有强大的操作系统所拥有的一切特点,包括多道程序、虚拟内存、开放性和兼容性、非常优秀的文件系统和目录系统等。

1.6.3　Linux操作系统

Linux是全世界影响力最大的免费使用和自由传播的类似于UNIX的操作系统,它的最初作者是芬兰学生Linux,得名于计算机业余爱好者Linus Torvalds。经过数十年全球各国编程爱好者的努力,Linux发展迅猛,成为除Windows以外使用最广的操作系统。

Linux主要用于基于x86系列CPU的计算机上。这个系统是由世界各地的成千上万的程序员设计和实现的。其目的是建立不受任何商品化软件的版权制约的、全世界都能自由使用的UNIX兼容产品。Linux属于自由软件,任何用户不用支付任何费用就可以获得该操作系统和它的源代码,并且可以根据资金的需要对它进行自由的拼装和必要的修改,无偿地使用它,无约束地继续传播。它具有UNIX操作系统的全部功能,任何会使用UNIX操作系统的用户不必重新学习和培训都可以直接使用Linux操作系统,想要学习UNIX操作系统的人也可以从学习Linux中得到帮助。Linux操作系统的整体设计是为了让操作系统在微处理机上更有效的运行。

Linux可以运行在各种硬件平台上。目前,Linux主要应用于网络服务器领域,因为它免费、开源、稳定、安全,非常适合网络应用。例如,现在最常见的建站程序PHP,就是基于

Linux 系统开发的语言。Linux 还应用于嵌入式系统，例如，机顶盒、移动电话等。

大多数普通用户可使用的 Linux 操作系统都是相关公司或爱好者基于 Linux 内核开发的 Linux 发行版。主流 Linux 发行版中通常有：RedHat（又称为“红帽”）、OpenLinux、SUSE、TurboLinux、Ubuntu 等。国内发行版中比较优秀的是“红旗 Linux”。

本章小结

操作系统已经发展了很多年，它有两个主要目的：第一，操作系统试图调度计算机活动，以确保计算机系统的高性能；第二，操作系统提供了一个环境，以便开发和运行程序。最初，计算机系统只能通过前端控制台来使用。汇编程序、装载程序、链接程序和编译程序这样一些软件改善了系统编程的方便性。

单道批处理系统通过使用驻留操作系统允许自动切换作业，进而大大地提高了计算机的整体利用率。计算机不再需要等待人工操作。但是，CPU 的利用率仍然很低。

为了改善计算机系统的整体性能，开发人员引入了多道程序设计的概念，这样多个作业可以同时位于内存中。通过在这些作业之间来回切换来提高 CPU 利用率，降低执行作业所需要的总时间。

多道程序设计也允许分时，分时操作系统允许多个用户同时交互地使用一个计算机系统。为了满足对交互时间的响应程度，计算机系统引入了实时的概念，它严格控制了程序执行的截止时间。

近年来，由于网络技术和硬件工艺的发展使得网络操作系统、多处理机和分布式操作系统得到发展和应用。智能终端设备的应用促进嵌入式操作系统的发展。

通过对本章的学习，读者应该对操作系统的历史、分类和基本概念有大致的了解。

习 题 1

1. 什么是操作系统？
2. 操作系统的目标有哪些？
3. 简述操作系统的发展过程。
4. 简述操作系统的类型。
5. 操作系统的基本特征有哪些？
6. 简述操作系统的主要功能。

第 2 章　进 程 管 理

在计算机操作系统中，进程是资源分配和独立运行的基本单位，也是一种研究操作系统的观点，因此，进程这一概念在操作系统中极为重要。本章主要介绍进程的定义、进程的状态及转换、进程控制、进程同步及线程。

2.1　进程的引入

早期的计算机系统属于单任务系统，一次只允许运行一道程序，多个程序以顺序方式依次运行。在现代计算机系统中，内存中通常存放多道程序，这些程序并发执行，为了描述并发程序执行时的特征，引入了进程。下面介绍程序的顺序执行和并发执行。

2.1.1　单道程序的顺序执行及特征

单道环境下，程序以顺序方式执行。

1. 程序的顺序执行

一个程序通常由若干个操作组成，这些操作必须按照某种先后次序执行，仅当前一个操作执行完成后才能执行后续操作；多个程序之间，仅当前一程序执行结束后才可执行下一程序，这类计算过程就是程序的顺序执行过程。

例如，系统中有 n 个作业，在执行每个作业时总是先输入程序和数据，然后进行计算，最后将所得的结果打印输出。若用 I_i、C_i 和 P_i 分别表示作业 i 的输入、计算及输出操作，则在顺序处理模式下这些作业操作的次序如图 2-1 所示。

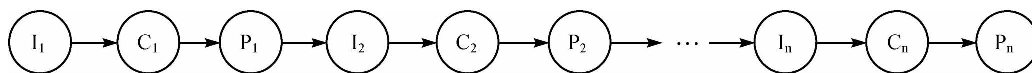

图 2-1　程序的顺序执行

图 2-1 用有向无环图来描述程序执行的先后次序，称为前驱图。图中的每个结点可以表示一条语句、一个程序段或一个进程，结点间的有向边表示两个结点之间存在的前驱关系。在前驱图中，没有前驱的结点称为初始结点，没有后继的结点称为终止结点。在图 2-1 中，I_1 为初始结点，P_n 为终止结点。

图中所描述的语句执行次序为：I_1 应首先执行，当 I_1 完成后才能执行 C_1，C_1 完成后才能执行 P_1，P_1 完成后执行 I_2，然后执行 C_2，C_2 完成后执行 P_2，以此顺序，最后执行 P_n。

2. 程序顺序执行时的特征

(1)顺序性。处理机的操作严格按照程序所规定的顺序执行，只有上一个操作完成后，下一个操作才能开始执行。

(2)封闭性。由于单道环境下只能运行一个程序,因此程序在运行时独占系统的全部资源,这些资源的状态只能由该程序改变,程序一旦开始运行,其执行结果不受外界因素影响。

(3)可再现性。只要程序执行时的初始条件和执行环境相同,当程序重复执行时,都将获得相同的结果,即程序的执行结果与时间无关。

2.1.2 多道程序的并发执行及特征

单道环境下,任何时刻系统中只能运行一道作业,系统的吞吐量和资源利用率较低。为提高计算机系统的吞吐量和资源利用率,现代计算机系统中采用了多道程序设计技术,使得系统内的多道程序可以并发执行。

1.程序的并发执行

对于图 2-1 的例子,在多道环境下,有些操作是可以并发执行的,如作业 1 的输入操作完成后即可以进行该作业的计算操作,与此同时可以进行作业 2 的输入操作,即作业 1 的计算操作和作业 2 的输入操作可以并发执行。图 2-2 给出了一批作业并发执行时的情况。

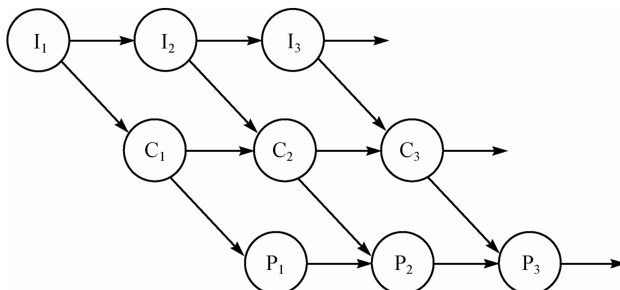

图 2-2 程序的并发执行

在图 2-2 中,I_1 先于 C_1 和 I_2,C_1 先于 P_1 和 C_2,P_1 先于 P_2;而 I_2 与 C_1,I_3、C_2 和 P_1 则可以并发执行。

程序的并发执行是指若干个程序或程序段同时在系统中运行,这些程序或程序段的执行在时间上是重叠的,即一个程序或程序段的执行尚未结束,另一个程序或程序段的执行已经开始。

思考:C_2 执行完成以后能立即执行 P_2 吗? 能立即执行 C_3 吗?

2.程序并发执行时的特征

程序的并发执行虽然提高了系统的处理能力和资源利用率,但它也带来了一些新问题,产生了一些与顺序执行时不同的特征:

(1)间断性。程序在并发执行时,由于它们共享资源或为完成同一项任务而相互合作,致使并发程序之间形成了相互制约的关系。例如,在图 2-2 中,若 C_1 未完成则不能进行 P_1,这是由相互合作完成同一项任务而产生的直接制约关系;若 I_1 未完成则不能进行 I_2,这是由共享资源而产生的间接制约关系。这种相互制约关系将导致并发程序具有"执行—暂停—执行"这种间断性的活动规律。

(2)失去封闭性。并发执行时,多个程序共享系统中的各种资源,因此这些资源的状态将由多个程序来改变,致使程序的运行失去封闭性。

(3)不可再现性。程序并发执行时,由于失去了封闭性,也将导致失去其执行结果的可

再现性。例如，有两个程序A和B，它们共享一个变量N。程序A对变量N执行N=N+1的操作；程序B对变量N执行N=0的操作。由于程序A和程序B都以各自独立的执行速度向前推进，故程序A的N=N+1操作既可以发生在程序B的N=0操作之前，也可以发生在N=0操作之后。假设某时刻N的值为5，对于上述两种情况，执行完程序A和B的相应语句后，得到的N值分别为0和1。

程序并发执行时具有结果不可再现的特征，这并不是使用者希望看到的结果。为此，要求程序在并发执行时必须保持封闭性和可再现性。由于并发执行失去封闭性的原因是共享资源，因此必须设法消去这种影响。

2.1.3　进程的概念

在多道程序环境下，程序的并发执行破坏了程序的封闭性和可再现性，为了正确的描述和控制程序的并发执行，操作系统中引入了一个新的概念——进程。

1.进程的定义

进程的概念是20世纪60年代初期，首先由麻省理工学院的Multics系统和IBM公司的Tss/360系统引入的。此后，有许多人对进程下过各式各样的定义，但直至目前还没有一个统一的定义，这里给出几种比较容易理解又能反映进程实质的定义：

(1)进程是程序在处理机上的一次执行过程。

(2)进程是可以和别的计算并行执行的计算。

(3)进程是程序在一个数据集合上的运行过程，是系统进行资源分配和调度的一个独立单位。

(4)进程是一个具有一定功能的程序关于某个数据集合的一次运行活动。

上述这些描述从不同的角度对进程进行了定义，尽管各有侧重，但它们在本质上是相同的。

综上所述，可将进程定义为：进程是进程实体的一次运行过程，是系统进行资源分配和调度的独立单位。

2.进程的特征

在多道程序系统中，多个进程并发执行，使得进程具有以下5个基本特征：

(1)动态性。进程是程序在处理机上的一次动态执行过程。动态特性还表现在它因创建而产生，由调度而执行，因得不到资源而暂停执行，最后因撤销而消亡。

(2)并发性。多个进程实体同时存在于内存中，在一段时间内都得到运行，它们在执行时间上是重叠的，即一个进程还未完成，另一个进程已开始执行。

(3)独立性。进程是能独立运行的基本单位，也是系统进行资源分配和调度的独立单位。

(4)异步性。系统中的各进程以独立的、不可预知的速度向前推进。

(5)结构性。为了描述和记录进程的动态变化过程，并使之能正确运行，为每个进程配置一个进程控制块(process control block，PCB)。从结构上看，每个进程都由程序段、数据段和一个进程控制块组成。程序段相当于做菜时的菜谱，描述了做菜的方法和步骤，数据段相当于用到的原料，而进程控制块相当于厨师按照菜谱将原料做成菜的过程。

2.2 进程的状态及组成

进程是程序的动态执行过程，为了描述进程的动态特征，使系统正确地控制进程的执行，将进程的生命周期划分为一组状态，用这些状态来描述进程的活动过程，并把进程的状态等信息记录在进程控制块中。

2.2.1 进程的基本状态

通常，一个进程至少应有就绪状态、执行状态和阻塞状态 3 种基本状态。

1. 进程的 3 种基本状态

(1)就绪状态。进程已获得了除处理机以外的所有资源，一旦获得处理机就可以立即执行，此时进程所处的状态为就绪状态。在操作系统中处于就绪状态的进程可以有多个，通常将它们排成一个队列，该队列称为就绪队列。

(2)执行状态。执行状态又称运行状态。当一个进程获得必要的资源并正在处理机上执行时，该进程所处的状态为执行状态。处于执行状态的进程数目不能大于处理机数目，在单处理机系统中处于执行状态的进程最多只有一个。

(3)阻塞状态。阻塞状态又称等待状态。正在执行的进程，由于发生某事件而暂时无法执行下去(如等待 I/O 完成)，此时进程所处的状态为阻塞状态。处于阻塞状态的进程尚不具备运行条件，这时即使处理机空闲，它也无法使用。系统中处于这种状态的进程可以有多个，这些进程排成一个队列，称为阻塞队列或等待队列，有些系统中根据阻塞的原因不同将阻塞进程排成多个队列。

注意：就绪和阻塞的区别是，就绪进程是因为其他条件都已具备，但缺少 CPU 而无法执行；阻塞进程是因为等待 CPU 之外的其他事件发生而暂时无法执行，由于其他条件不具备，即使 CPU 空闲，进程也无法执行。

2. 进程的 3 种基本状态的转换及原因

随着进程自身的推进和系统环境的变化，进程的状态在不断地发生变化。进程的 3 种基本状态间的转换及原因如下：

(1)就绪→执行。处于就绪状态的进程，当进程调度程序为其分配了处理机后，该进程便由就绪状态转变为执行状态。

(2)执行→阻塞。正在执行的进程因等待某事件发生，如进程提出 I/O 请求并等待 I/O 操作完成时，则进程由执行状态转变为阻塞状态。

(3)阻塞→就绪。处于阻塞状态的进程，当其等待的事件已经完成时，如 I/O 操作完成，则进程由阻塞状态转变为就绪状态。

(4)执行→就绪。正在执行的进程，如因时间片完而暂停执行，该进程便由执行状态转变为就绪状态。

进程的状态转换可以用如图 2-3 所示的进程状态转换图来描述。

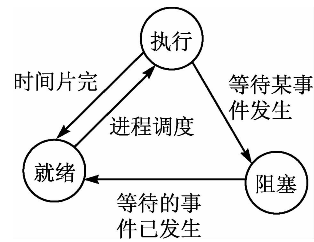

图 2-3 进程状态转换

2.2.2 进程的挂起状态

在某些系统中，为了更好地管理和调度进程及适应系统的功能目标，引入了挂起状态。

1.引入挂起状态的原因

引入挂起状态可能基于下述原因：

(1)系统有时可能出故障或某些功能受到破坏，这时就需要暂时将系统中的进程挂起，以便系统故障消除后，再将这些进程恢复到原来状态。

(2)用户检查自己作业的中间执行情况和中间结果时，因同预期想法不符而产生怀疑，这时用户要求挂起进程，以便进行某些检查和改正。

(3)系统中有时负荷过重(进程数过多)，资源数相对不足，从而造成系统效率下降。此时需要挂起一部分进程以调整系统负荷，等系统中负荷减轻后再恢复被挂起进程的执行。

(4)在操作系统中引入了虚拟存储管理技术后，需要区分进程是驻留在内存还是外存，此时可以用挂起表示驻留在外存。

2.具有挂起状态的进程状态转换

引入挂起状态以后，进程增加了两个新的状态：挂起就绪和挂起阻塞。为了易于区分，将原来的就绪状态称为活动就绪，将原来的阻塞状态称为活动阻塞。

图 2-4 所示是具有挂起状态的进程状态转换图。

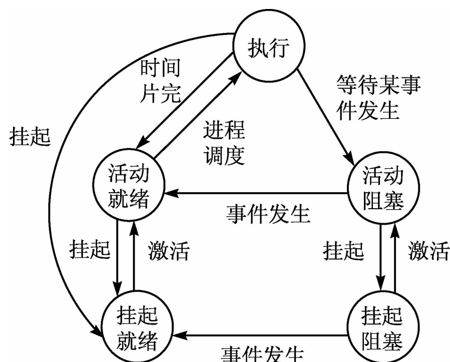

图 2-4 具有挂起状态的进程状态转换

从图 2-4 中可以看出，处于执行状态或活动就绪状态的进程，因挂起而变为挂起就绪状态，处于挂起就绪状态的进程不能参与争夺处理机，即进程调度程序不会调度处于挂起就绪状态的进程；处于挂起就绪状态的进程因激活而变为活动就绪状态，此时可以竞争处理机；

处于活动阻塞状态的进程因挂起而变为挂起阻塞状态；处于挂起阻塞状态的进程因激活而变为活动阻塞状态；处于挂起阻塞状态的进程，其所等待的事件发生后，该进程就由原来的挂起阻塞状态变为挂起就绪状态。

2.2.3 进程控制块

进程控制块 PCB 是进程实体的一部分，是进程存在的唯一标志。为了准确地描述每个进程，并对进程进行有效的控制与管理，系统为每个进程创建了一个进程控制块。系统通过 PCB 感知进程的存在，通过 PCB 中各项变量的变化了解进程的执行状况，根据进程控制块中各项信息对进程进行调度、控制和管理。因此，当进程创建时，系统为它建立一个 PCB；当进程在执行中状态发生变化时，系统将其执行信息记录在 PCB 中；当进程执行完毕时，系统回收其 PCB。所以 PCB 是进程在其生命周期中的管理档案。尽管不同操作系统中 PCB 的结构不同，但通常包括下面所列出的内容：

(1)进程标志符。进程标志符分为内部标志符和外部标志符。内部标志符是标志一个进程的唯一的整数，以区别于系统内部的其他进程。在进程创建时，由系统为进程分配唯一的内部标志符。内部标志符主要是用来方便系统管理的。外部标志符是由创建者提供的字母、数字组成的标志符，是用户访问进程时使用的。

(2)进程当前状态。进程当前状态说明进程当前所处的状态，以作为进程调度程序分配处理机的依据。

(3)进程队列指针。进程队列指针用于记录 PCB 队列中下一个 PCB 的地址。系统中的 PCB 可能组织成多个队列，如就绪队列、阻塞队列等。

(4)程序和数据地址。程序和数据地址指进程的程序和数据在内存或外存中的存放地址。

(5)进程优先级。进程优先级反映进程获得 CPU 的优先级别，优先级高的进程可以优先获得处理机。

(6)CPU 现场保护区。当进程因某种原因释放处理机时，CPU 现场信息被保存在 PCB 的该区域中，以便在进程重新获得处理机后能恢复执行。通常被保护的信息有通用寄存器、程序计数器和程序状态字等内容。

(7)通信信息。通信信息记录进程在执行过程中与其他进程所发生的信息交换情况。

(8)家族关系。有的系统允许进程创建子进程，从而形成一个进程家族树。在 PCB 中必须指明本进程与家族的关系，如它的子进程与父进程的标志。

(9)资源清单。资源清单列出进程所需资源及当前已分配资源。

在一个系统中，通常存在着许多进程，它们有的处于就绪状态，有的处于阻塞状态，而且阻塞的原因各不相同。为了方便进程的调度和管理，需要将各进程的 PCB 用适当的方法组织起来。目前常用的组织方式有链接方式和索引方式两种。链接方式是将同一状态的 PCB 链接成一个队列，不同状态对应多个不同的队列，如就绪队列和阻塞队列等。索引方式是将同一状态的进程组织在一个索引表中，索引表的表项指向相应的进程控制块，不同状态对应不同的索引表，如就绪索引表和阻塞索引表等。

2.3　进程控制

进程控制的职责是对系统中的所有进程实施有效的管理，其功能包括进程创建、进程撤销、进程阻塞与唤醒等。进程控制是通过执行各种原语来实现的。原语是由若干条机器指令构成的一段程序，用以完成特定功能，这段程序在执行期间不可分割。这就是说，原语的执行不能被中断，所以原语操作具有原子性。进程控制功能一般由操作系统的内核来完成。

2.3.1　操作系统内核

在现代操作系统设计中，往往把一些与硬件紧密相关的模块或运行频率较高的模块以及为许多模块所公用的一些基本操作安排在靠近硬件的软件层次中，并使它们常驻内存，以提高操作系统的运行效率，通常把这部分软件称为操作系统内核。操作系统内核的主要功能包括中断、时钟管理、进程管理、存储器管理、设备管理等。

任何一个计算机系统中都有两种运行状态，即核心态和用户态。当操作系统内核程序执行时处于核心态，当用户程序执行时处于用户态。

(1)核心态又称管态、系统态，是操作系统管理程序执行时计算机所处的状态。这种状态具有较高的特权，能执行一切指令，访问所有的寄存器和存储区。

(2)用户态又称目态，是用户程序执行时计算机所处的状态。这种状态具有较低的特权，只能执行规定的指令，访问指定的寄存器和存储区。

2.3.2　进程的创建与撤销

1.进程家族树

一个进程可以创建若干个新进程，新创建的进程又可以创建子进程。为了描述进程之间的创建关系，引入了进程家族树。进程家族树是描述进程家族关系的一棵有向树。树中的结点表示进程，若进程 A 创建了进程 B，则从结点 A 有一条有向边指向结点 B，说明进程 A 是进程 B 的父进程，进程 B 是进程 A 的子进程。创建父进程的进程称为祖父进程，从而形成了一棵进程家族树，把树的根结点称为进程家族的祖先。例如，若进程 A 创建了子进程 B、C、D，进程 B 又创建了自己的子进程 E，进程 C 创建了子进程 F，则构成了一棵如图 2-5 所示的进程家族树，其中进程 A 是该进程家族的祖先。

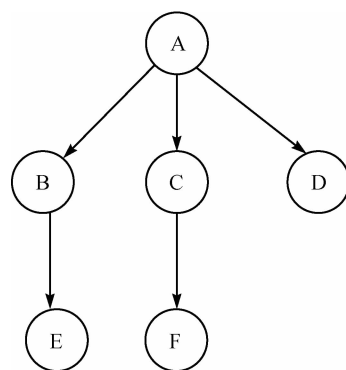

图 2-5　进程家族树

2.进程创建原语

在多道程序环境中，为了使一个程序能运行，必须为它创建进程。引起进程创建的事件大致有以下几类：

(1)作业调度。在批处理系统中，提交给操作系统的作业通常存放在磁带或磁盘上。当作业调度程序选中某个作业时，便为该作业创建进程，分配必要的资源并插入就绪队列。

(2)用户登录。在交互式系统中,当用户登录进入系统时,操作系统要建立新进程(如命令解释进程),负责接收并解释用户输入的命令。

(3)操作系统提供服务。当运行中的用户程序向操作系统提出某种请求时,操作系统也会创建进程来完成用户程序所需要的服务功能。例如,用户程序请求打印一个文件,操作系统将建立一个打印进程,负责管理用户程序需要的打印工作。

(4)应用请求。前3种情况都是由操作系统根据需要为用户创建进程的,事实上应用程序也可以根据需要来创建一个新进程,使之与父进程并发执行,以完成特定的任务。

进程创建原语的功能是创建一个新进程,其主要操作过程如下:

(1)向系统申请一个空闲PCB。从系统的PCB表中找出一个空闲的PCB表项,并指定唯一的进程标志号。

(2)为新进程分配资源。根据新进程提出的资源需求为其分配资源,例如,为新进程分配内存空间以存放程序及数据。

(3)初始化新进程的PCB。在新进程的PCB中填入进程名、家族信息、程序和数据地址、进程优先级、资源清单及进程状态等信息。一般新进程的状态为就绪状态。

(4)将新进程的PCB插入就绪队列。

3.进程撤销原语

一个进程在完成其任务后应予以撤销,以便及时释放它所占有的各类资源。引起进程撤销的事件大致有以下几类:

(1)进程正常结束。当一个进程完成其任务后,应该将其撤销并释放其所占有的资源。

(2)进程异常结束。在进程运行期间,如果出现了错误或故障,则进程被迫结束运行。导致进程异常结束的事件较多,如运行超时、内存不足、越界错误、I/O故障、算术运算错误等。

(3)外界干预。进程因外界的干预而被迫结束运行。外界干预包括操作人员或操作系统的干预,如为了解除死锁,操作人员或操作系统要求撤销进程;当父进程终止时操作系统会终止其子孙进程;父进程有权请求系统终止其子孙进程。

撤销原语的功能是撤销一个进程,其主要操作过程如下:

(1)从系统的PCB表中找到被撤销进程的PCB。

(2)检查被撤销进程的状态是否为执行状态,若是则立即停止该进程的执行,设置重新调度标志,以便在该进程撤销后将处理机分配给其他进程。

(3)检查被撤销进程是否有子孙进程,若有子孙进程还应撤销该进程的子孙进程。

(4)回收该进程占有的全部资源并回收其PCB。

2.3.3 进程阻塞与唤醒

当进程在执行过程中因等待某事件的发生而暂停执行时,进程调用阻塞原语将自己阻塞起来,并主动让出处理机。当阻塞进程等待的事件发生时,由事件的发现者进程调用唤醒原语将阻塞的进程唤醒,使其进入就绪状态。

1.引起进程阻塞与唤醒的事件

引起进程阻塞和唤醒的事件大致有以下几类:

(1)请求系统服务。当正在执行的进程向系统请求某种服务时,由于某种原因其要求无

法立即满足，进程便暂停执行而变为阻塞状态。例如，当进程在执行中请求打印服务时，由于打印机已被其他进程占用，请求者只能进入阻塞状态去等待。当进程请求的系统服务完成时，应将阻塞进程唤醒。

(2)启动某种操作并等待操作完成。当进程执行时启动了某操作，且进程只有在该操作完成后才能继续执行，那么进程也将暂停执行而变为阻塞状态。例如，进程启动了某I/O设备进行I/O操作，但由于设备速度较慢而不能立刻完成指定的I/O任务，所以进程进入阻塞状态等待。当进程启动的操作完成时，应将阻塞进程唤醒。

(3)等待合作进程的协同配合。相互合作的进程，有时需要等待合作进程提供新的数据或等待合作进程作出某种配合而暂停执行，那么进程也将停止执行而变为阻塞状态。例如，计算过程不断地计算结果并存入缓冲区中，而打印进程不断地从缓冲区中取出数据进行打印。如果计算进程尚未将数据送到缓冲区中，则打印进程只能变为阻塞状态去等待。当合作进程完成协同任务时，应将阻塞进程唤醒。

(4)系统进程无新工作可做。系统中往往设置了一些具有特定功能的系统进程，每当它们的任务完成后便将自己阻塞起来，以等待新任务的到来。例如，系统中设置的发送进程，若已有发送请求全部完成且尚无新的发送请求，这时发送进程将阻塞等待。当系统进程收到了新的任务请求时，应将阻塞进程唤醒。

2. 进程阻塞原语

阻塞原语的功能是将进程由执行状态转变为阻塞状态，其主要操作过程如下：

(1)停止当前进程的执行。进程阻塞时，由于该进程正处于执行状态，故应停止该进程的执行。

(2)保存该进程的CPU现场信息。为了使进程以后能够重新调度执行，应将该进程的现场信息送入其PCB现场保护区中保存起来。

(3)将进程状态改为阻塞，并插入到相应事件的等待队列中。

(4)转进程调度程序，从就绪队列中选择一个新的进程执行。

3. 进程唤醒原语

唤醒原语的功能是将进程由阻塞状态转变为就绪状态，其主要操作过程如下：

(1)将被唤醒进程从相应的等待队列中移出。

(2)将进程状态改为就绪，并将该进程插入就绪队列。

(3)在某些系统中，如果被唤醒进程比当前执行进程的优先级更高，可能需要设置调度标志，重新调度。

应当注意的是，一个进程由执行状态转变为阻塞状态，是这个进程自己调用阻塞原语去完成的，而进程由阻塞状态转变为就绪状态，则是其他进程(一般是系统进程或被唤醒进程的合作进程)调用唤醒原语实现的。

2.3.4 进程的挂起与激活

当需要挂起某个进程时可以调用挂起原语，需要激活某个进程时可以调用激活原语。

1. 进程挂起原语

挂起原语使进程由活动状态变成挂起状态。挂起的主要操作过程如下：

(1)以进程标志符为索引，到PCB表中查找该进程的PCB。

(2)检查该进程的状态。

(3)若状态为执行,则停止该进程执行并保护 CPU 现场信息,将该进程状态改为挂起就绪;若状态为活动阻塞,则将该进程状态改为挂起阻塞;若状态为活动就绪,则将该进程状态改为挂起就绪。

(4)若进程挂起前为执行状态,则转进程调度,从就绪队列中选择一个新进程投入运行。

2. 进程激活原语

激活原语使处于挂起状态的进程变成活动的。激活原语的主要操作过程如下:

(1)以进程标志符为索引,到 PCB 表中查找该进程的 PCB。

(2)检查该进程的状态。若状态为挂起阻塞,则将该进程状态改为活动阻塞。若状态为挂起就绪,则将该进程状态改为活动就绪。

(3)若进程激活后为活动就绪状态,可能需要转进程调度。

2.4 进 程 同 步

进程同步的主要任务是对多个相关进程在执行次序上进行协调,以使并发执行的诸进程之间能有效地共享资源和相互合作,从而使程序的执行具有可再现性。

2.4.1 进程同步的基本概念

1. 进程间的制约关系

在多道程序环境下,当程序并发执行时,由于资源共享和进程合作,使同处于一个系统中的诸进程之间可能存在着以下两种形式的制约关系。

(1)间接制约关系。间接制约是指由于多个并发执行的进程共享系统资源,使得它们必须按照一定的次序来使用资源。例如,有两个进程 A 和 B 共享打印机,如果进程 A 正在使用打印机,进程 B 提出打印请求,此时进程 B 必须阻塞,当进程 A 打印完毕释放打印机后进程 B 才可获得打印机。

(2)直接制约关系。直接制约是指由于进程间相互合作,使得各进程必须相互协调、按照一定的次序向前推进。例如,有一数据采集进程和一计算进程相互合作完成数据处理。数据采集进程将采集的数据写入缓冲区,计算进程取出数据进行计算。当该缓冲区空时,计算进程因不能获得所需数据而阻塞,当数据采集进程把数据写入缓冲区后,便将计算进程唤醒;反之,当缓冲区已满时,数据采集进程因不能再向缓冲区写入数据而阻塞,当计算进程将缓冲区数据取走后便可唤醒数据采集进程。

2. 临界资源与临界区

临界资源是指在一段时间内只允许一个进程访问的资源。只有在占用临界资源的进程释放临界资源后,其他进程才允许访问。也就是说,临界资源是排他性资源,必须采取互斥的方式访问。系统中很多物理设备,如磁带机、打印机等都属于临界资源,一些软件中使用的变量、栈和队列等数据结构也属于临界资源。

进程中访问临界资源的那段代码称为进程的临界区。进程对临界资源必须采取互斥的方式访问，也就是说，进程必须互斥地进入自己的临界区。

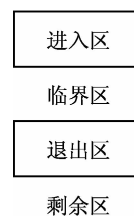

图 2-6　进程对临界资源的访问过程

为使多个进程能互斥地访问某临界资源，每个进程在进入临界区之前，应先对临界资源进行检查，看它是否正被访问。如果此刻该临界资源未被访问，进程便可进入临界区对该资源进行访问，并设置它正被访问的标志；如果此刻该临界资源正被某进程访问，则该进程不能进入临界区。这段用于检查临界资源使用状态的代码称为进入区。相应地，访问临界资源结束后，还应将临界区正被访问的标志恢复为未被访问的标志，使得其他进程可以访问临界资源。这段代码称为退出区。进入区、临界区和退出区之外的代码称为剩余区。因此一个进程对临界资源的访问过程可分为如图 2-6 所示的 4 部分。

3. 临界区管理的准则

为了合理利用临界资源，保证进程互斥地进入临界区，对临界区的管理应遵循以下准则：

(1)空闲让进。当无进程处于临界区时，表明临界资源处于空闲状态，应允许一个请求进入临界区的进程立即进入自己的临界区，以有效地利用临界资源。

(2)忙则等待。当已有进程进入临界区时，表明临界资源正在被访问，因而其他试图进入临界区的进程必须等待，以保证对临界资源的互斥访问。

(3)有限等待。对要求访问临界资源的进程，应保证在有限时间内能进入自己的临界区，以免陷入“死等”状态。

(4)让权等待。当进程不能进入自己的临界区时，应立即释放处理机，以免进程陷入“忙等”状态。

2.4.2　信号量机制

1965 年，荷兰学者 Dijkstra 提出的信号量(semaphore)机制是一种卓有成效的进程同步工具。在长期且广泛的应用中，信号量机制又得到了很大的发展，它从整型信号量经记录型信号量，进而发展为信号量集机制。现在，信号量机制已被广泛地应用于单处理机和多处理机系统以及计算机网络中。在此，介绍记录型信号量和 P、V 操作。

1. 记录型信号量

在记录型信号量机制中，除了一个用于代表资源数目的整型变量 value 外，还有一个进程链表指针 L，用于链接所有请求该资源的等待进程。记录型信号量可描述为：

```
typedef struct{
    int value;
    Queue_PCB L;
}semaphore;
```

2. P、V 操作

除了进行初始化以外，只能通过两个原子操作(wait 和 signal)来访问记录型信号量，其他访问方式都是不允许的，通常称这两个操作为 P 操作和 V 操作。

P 操作的步骤为：

(1)信号量值减 1。

(2)若信号量值小于0,则将进程插入到该信号量的等待队列中。

V操作的步骤为:

(1)信号量值加1。

(2)若信号量值小于等于0,则从该信号量的等待队列中唤醒第一个进程。

P、V操作可描述为:

```
P(semaphore S)
{
    S.value=S.value-1;
    if(S.value<0) block(S.L);
}
V(semaphore S)
{
    S.value=S.value+1;
    if(S.value<=0) wakeup(S.L);
}
```

P操作每次对信号量的值减1,表示进程请求一个资源或检查某条件是否满足,无可用资源或条件不满足时阻塞进程;V操作每次对信号量的值加1,表示进程释放一个资源或给其他进程发信号,有阻塞进程时可以唤醒一个进程。

3.信号量的物理意义

信号量的物理意义为:

(1)S.value的初值表示系统中某类资源的数目,因而又称为资源信号量;当S.value初值为1时,表示只允许一个进程访问临界资源,此时的信号量转化为互斥信号量,用于进程互斥。

(2)当S.value≥0时,表示该类资源的可用数目。

(3)当S.value<0时,|S.value|表示在该信号量链表中已阻塞进程的数目。

2.4.3 信号量的应用

记录型信号量既可以实现进程互斥,也可以实现进程同步,而实现前驱关系是进程同步中较简单的一种,下面将从实现进程互斥和实现前驱关系两方面来说明信号量的使用。

1.利用信号量实现进程互斥

为使多个进程互斥地进入临界区,只需为该资源设置一互斥信号量mutex,并设其初始值为1,然后将各进程访问该资源的临界区置于P(mutex)和V(mutex)操作之间即可。这样,每个欲访问该临界资源的进程在进入临界区之前,都要先对mutex执行P操作,若该资源此刻未被访问,本次P操作必然成功,进程便可进入自己的临界区,这时若有其他进程也欲进入自己的临界区,由于对mutex执行P操作,信号量的值小于0,因而该进程阻塞,从而保证了该临界资源能被互斥的访问。当访问临界资源的进程退出临界区后,又对mutex执行V操作,以便释放该临界资源。利用信号量实现进程互斥的进程可描述如下:

```
semaphore mutex=1;
main( )
```

```
{
    cobegin
        P1( );
        P2( );
    coend
}
P1( )
{
    P(mutex);
    critical section
    V(mutex);
    remainder section
}
P2( )
{
    P(mutex);
    critical section
    V(mutex);
    remainder section
}
```

在利用信号量机制实现进程互斥时应注意，P(mutex)和 V(mutex)必须成对地出现。缺少 P(mutex)将会导致系统混乱，不能保证对临界资源的互斥访问；而缺少 V(mutex)将会使临界资源永远不被释放，从而使因等待该资源而阻塞的进程不能被唤醒。

2. 利用信号量实现前驱关系

还可利用信号量来描述程序或语句之间的前驱关系。如图 2-7 所示的前驱图中，P1、P2、P3、…、P7 是一组相互合作的进程。可以利用信号量，按照进程间的前驱关系，写出一个可并发执行的程序。

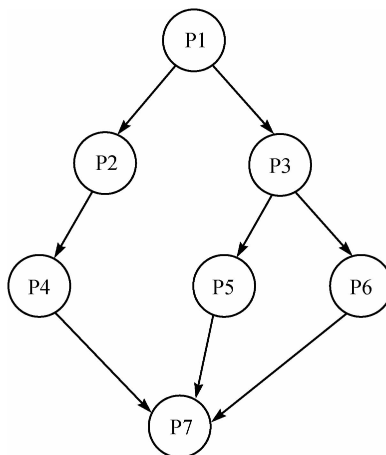

图 2-7 前驱图

分析：根据图 2-7 所示，进程间的前驱关系为：

P1 先执行，然后 P2 和 P3 执行，P2 完成后 P4 执行，P3 完成后 P5 和 P6 开始执行，P4、P5 和 P6 都完成后执行 P7。

为保证 P2 在 P1 完成后才开始执行，P2 执行前需先检查 P1 是否完成，若 P1 完成则 P2 执行，否则 P2 阻塞，直到 P1 完成后将其唤醒。实现 P1 和 P2 之间的前驱关系，只需为 P1 和 P2 设一个信号量 a，初始值为 0。在 P1 之后执行 V(a)，在 P2 之前先执行 P(a)。这样，如果 P2 先执行，P(a)操作使 a 的值小于 0，因此 P2 会阻塞。P1 执行后会执行 V(a)，使 a 的值加 1，因此会唤醒被阻塞的进程。

根据上述分析，应设信号量 a、b、c、d、e、f、g、h，初始值都为 0。

这 7 个进程的同步程序为：

```
semaphore a=0, b=0, c=0, d=0, e=0, f=0, g=0, h=0;
main()
{
    cobegin
        P1( );
        P2( );
        P3( );
        P4( );
        P5( );
        P6( );
        P7( );
    coend
}
P1( )
{
     ⋮
    V(a);
    V(b);
}
P2( )
{
    P(a);
     ⋮
    V(c);
}
P3( )
{
    P(b);
     ⋮
    V(d);
```

```
    V(e);
}
P4( )
{
    P(c);
     ⋮
    V(f);
}
P5( )
{
    P(d);
     ⋮
    V(g);
}
P6( )
{
    P(e);
     ⋮
    V(h);
}
P7( )
{
    P(f);
    P(g);
    P(h);
     ⋮
}
```

2.4.4　管程机制

虽然信号量机制是一种既方便又有效的进程同步机制,但大量的P、V操作分散在各个进程中,不仅管理麻烦,还会因同步操作的使用不当而导致系统死锁。在解决上述问题的过程中,产生了一种新的进程同步工具——管程(monitor)。

1.管程的定义

代表共享资源的数据结构,以及由对该共享数据结构实施操作的一组过程所组成的资源管理程序,共同构成了一个操作系统的资源管理模块,称之为管程。管程被请求和释放资源的进程所调用。

Hansen为管程所下的定义是:"一个管程定义了一个数据结构和能为并发进程所执行(在该数据结构上)的一组操作,这组操作能同步进程和改变管程中的数据。"

管程由4部分组成:管程的名称、局部于管程内部的共享数据结构说明、对该数据结构

进行操作的一组过程和对局部于管程内部的共享数据设置初始值的语句。管程的语法描述如下：

```
MONITOR monitor_name;
<共享数据结构说明>;
<过程名>(<形式参数表>)
{
 ⋮
}
 ⋮
<管程的局部数据初始化语句序列>;
```

局部于管程内部的数据结构，仅能被局部于管程内部的过程所访问，任何管程外的过程都不能访问它；反之，局部于管程内部的过程也仅能访问管程内的数据结构。由此可见，管程相当于围墙，它把共享数据结构和对它进行操作的若干过程围了起来，所有进程要访问临界资源时，都必须经过管程（相当于通过围墙的门）才能进入，而管程每次只准许一个进程进入管程，从而实现了进程互斥。

2. 条件变量

在利用管程实现进程同步时，必须设置同步工具，如两个同步操作原语 wait 和 signal。当某进程通过管程请求获得临界资源而未能满足时，管程便调用 wait 原语使该进程等待，并将其排在等待队列上，仅当另一进程访问完成并释放该资源之后，管程才又调用 signal 原语，唤醒等待队列中的队首进程。

但是仅有上述的同步工具是不够的。当一个进程调用了管程，在管程中被阻塞或挂起，直到阻塞或挂起的原因解除，而在此期间，如果该进程不释放管程，则其他进程无法进入管程，被迫长时间地等待。为了解决这个问题，引入了条件变量 condition。通常，一个进程被阻塞或挂起的原因可以有多个，因此在管程中设置了多个条件变量，对这些条件变量的访问，只能在管程中进行。

管程中对每个条件变量都需予以说明，其形式为“condition x，y；”，对条件变量的操作仅是 wait 和 signal，因此条件变量也是一种抽象数据类型，每个条件变量保存了一个链表，用于记录因该条件变量而阻塞的所有进程，同时提供的两个操作可表示为 x. wait 和 x. signal，其含义为：

（1）x. wait：正在调用管程的进程因 x 条件需要被阻塞或挂起，则调用 x. wait 将自己插入到 x 条件的等待队列上，并释放管程，直到 x 条件变化。此时其他进程可以使用该管程。

（2）x. signal：正在调用管程的进程发现 x 条件发生了变化，则调用 x. signal，重新启动一个因 x 条件而阻塞或挂起的进程。如果存在多个这样的进程，则选择其中的一个，如果没有，则继续执行原进程，而不产生任何结果。这与信号量机制中的 signal 操作不同，因为后者总是要执行 S＝S＋1 的操作，因而总会改变信号量的状态。

2.5 经典进程的同步问题

在多道程序环境下，进程同步问题十分重要，也是相当有趣的问题，因而吸引了不少学

者对它进行研究,由此而产生了一系列经典的进程同步问题,其中较有代表性的是“哲学家进餐问题”、“生产者—消费者问题”和“读者—写者问题”等。

2.5.1 哲学家进餐问题

哲学家进餐问题是典型的同步问题。该问题是描述 5 个哲学家共用 1 张圆桌,分别坐在周围的 5 张椅子上,在圆桌上有 5 个碗和 5 支筷子,他们的生活方式是交替地进行思考和进餐。平时,哲学家进行思考,饥饿时便试图取用其左右最靠近他的筷子,只有在他拿到两支筷子时才能进餐。进餐完毕,放下筷子继续思考。

分析:放在桌子上的筷子是临界资源,在一段时间内只允许一位哲学家使用。为了实现对筷子的互斥使用,可以为每支筷子设一个互斥信号量,由这 5 个信号量构成信号量数组。这 5 个哲学家的同步算法描述如下:

```
semaphore chopstick[5] ={1,1,1,1,1};
main()
{
    cobegin
        philosopher(0);
        philosopher(1);
        philosopher(2);
        philosopher(3);
        philosopher(4);
    coend
}
philosopher(int i)
{
    while(1)
    {
        think;
        P(chopstick[i]);
        P(chopstick[(i+1) mod 5]);
        eat;
        V(chopstick[i]);
        V(chopstick[(i+1) mod 5]);
    }
}
```

实现互斥时需要注意的问题:

(1)实现一种资源的互斥只需设一个互斥信号量,初始值为 1。

(2)对互斥信号量的 P、V 操作必须成对出现,且出现在同一个程序中。若上述算法中缺少一个 P 操作,会导致相邻哲学家同时拿起一支筷子,即不能保证互斥;若缺少一个 V 操作,即哲学家用完筷子未释放,将使该筷子永远无法再被使用,从而造成一些哲学家因为拿不到筷子而被“饿死”。

上述解法可保证不会有两个相邻的哲学家同时进餐,但有可能引起死锁。假如 5 位哲学家同时饥饿而各自拿起左边的筷子时,就会使 5 个信号量均为 0;当他们再试图去拿右边的筷子时,都将因无筷子可拿而无限期地等待。对于这样的死锁问题,可采取以下几种解决方法:

(1)至多只允许有 4 位哲学家同时去拿左边的筷子,最终能保证至少有一位哲学家能够进餐,并在用毕释放出他用过的两支筷子,从而使更多的哲学家能够进餐。

(2)规定奇数号哲学家先拿他左边的筷子,然后再去拿右边的筷子,而偶数号哲学家则相反。按此规定,将是 1、2 号哲学家竞争 2 号筷子,3、4 号哲学家竞争 4 号筷子,即 5 位哲学家都先竞争奇数号筷子,获得后,再去竞争偶数号筷子,必然会有一位哲学家能获得两支筷子而进餐。

2.5.2 生产者—消费者问题

生产者—消费者问题是一种常见的同步问题。比如,计算进程和打印进程之间,计算进程为打印进程提供数据,计算进程为生产者,打印进程为消费者。

1.一个生产者和一个消费者问题

一个生产者和一个消费者,共享一个缓冲区。生产者不断生产物品,每生产一件便存入缓冲区,消费者要不断从缓冲区中取出一件物品消费。缓冲区只能容纳一件物品,生产者要等消费者取走物品后才能放入下一件物品,而消费者取走一件物品后要等生产者放入下一件物品才能再取。如图 2-8 所示。

图 2-8 一个生产者和一个消费者问题

分析:生产者向缓冲区中放物品之前,应检查缓冲区是否已空,若缓冲区不空,生产者等待,否则投放物品,因此,应给生产者设一个信号量,用于表示空缓冲区的数量。消费者从缓冲区中取物品之前应检查缓冲区中是否有产品,若缓冲区空,消费者等待,否则取走产品。因此,应给消费者设一个信号量,用于表示满缓冲区的数量。另外,生产者放入物品以后应唤醒等待的消费者;而消费者取走物品以后应唤醒等待的生产者。生产者和消费者之间只存在同步关系。因此只需设同步信号量。

同步算法:

```
semaphore empty=1, full=0;
main()
{
    cobegin
        producer( );
        consumer( );
    coend
}
producer( )
```

```
{
    while(1)
    {
        produce a product nextp;
        P(empty);
        buffer=nextp;
        V(full);
    }
}
consumer( )
{
    while(1)
    {
        P(full);
        nextc=buffer;
        V(empty);
        consume the product;
    }
}
```

实现同步时需要注意的问题：

(1)设资源信号量，往往多于一个。

(2)对每个信号量的P和V操作成对调用，且出现在不同的程序中。

(3)至少有一个信号量的初始值等于1，否则所有进程都无法执行。

2.一群生产者和一群消费者问题

有一群生产者进程在生产产品，并将这些产品提供给消费者进程去消费。为使生产者进程与消费者进程能并发执行，在两者之间设置了一个具有n个缓冲区的循环缓冲池，生产者进程将它所生产的产品放入一个缓冲区中；消费者进程可从一个缓冲区中取走产品去消费。不允许消费者进程到一个空缓冲区去取产品，也不允许生产者进程向一个已装满产品且尚未被取走的缓冲区中投放产品，如图2-9所示。

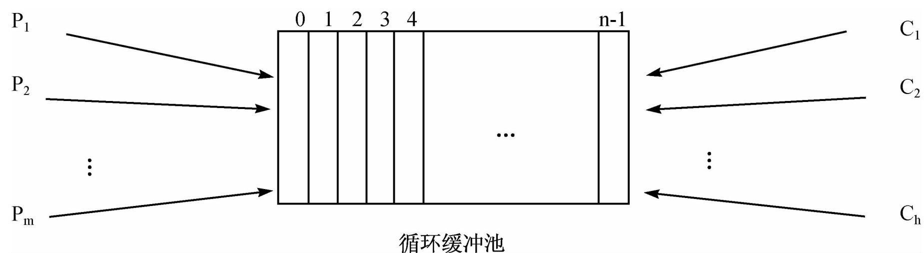

图2-9 一群生产者和一群消费者问题

分析：缓冲池不满时，生产者才可以放产品，否则，生产者要等待消费者取走产品后才能放产品；缓冲池不空时，消费者才可以取产品，否则，消费者要等待生产者投放产品后才能取产品。因此，生产者和消费者之间存在同步关系。

由于缓冲池被所有的生产者和消费者共同访问，因此缓冲池是临界资源，必须互斥访问。因此，所有的生产者和消费者之间存在互斥关系。

设互斥信号量 mutex 实现诸进程对缓冲池的互斥使用。设信号量 empty 和 full 分别表示缓冲池中空缓冲区和满缓冲区的数量。一群生产者和一群消费者问题可描述如下：

```
semaphore mutex=1, empty=n, full=0;
int in=0, out=0;
main()
{
    cobegin
        producer( );
        consumer( );
    coend
}
producer( )
{
    while(1)
    {
        produce a product nextp;
        P(empty);
        P(mutex);
        buffer[in]=nextp;
        in=(in+1) mod n;
        V(mutex);
        V(full);
    }
}
consumer( )
{
    while(1)
    {
        P(full);
        P(mutex);
        nextc=buffer[out];
        out=(out+1) mod n;
        V(mutex);
        V(empty);
        consume the product nextc;
    }
}
```

需要注意的是，在每个程序中的多个 P 操作顺序不能颠倒，应先执行对资源信号量的 P

操作，然后再执行对互斥信号量的P操作，否则可能引起进程死锁。例如，将生产者算法中的P操作颠倒顺序，那么缓冲池满了以后，生产者再次投放产品时，执行P(mutex)使得mutex值为0，该生产者继续执行P(empty)时由于无空缓冲区将被阻塞，只有消费者取产品后才会被唤醒；而消费者取产品时执行P(mutex)将被阻塞，等待生产者释放缓冲池时才会被唤醒。因此，生产者和消费者之间相互等待，都无法继续执行，最终导致死锁产生。

生产者和消费者访问缓冲池时要互斥，原因在于多个生产者共享一个变量in，有可能向同一个空的缓冲区中投放产品，多个消费者共享一个变量out，有可能从同一个满的缓冲区中取产品。因此，只要生产者投放产品时彼此互斥，消费者取产品时彼此互斥即可。该问题可以通过给生产者和消费者分别设互斥信号量mutex1和mutex2来实现。只需分别将生产者和消费者算法中对mutex的P、V操作改为对mutex1的P、V操作和对mutex2的P、V操作即可。同步算法如下：

```
semaphore mutex1=1, mutex2=1, empty=n, full= 0;
int in=0, out=0;
main()
{
    cobegin
        producer( );
        consumer( );
    coend
}
producer( )
{
    while(1)
    {
        produce a product nextp;
        P(empty);
        P(mutex1);
        buffer[in]=nextp;
        in=(in+1) mod n;
        V(mutex1);
        V(full);
    }
}
consumer( )
{
    while(1)
    {
        P(full);
        P(mutex2);
        nextc=buffer[out];
```

```
            out=(out+1) mod n;
            V(mutex2);
            V(empty);
            consume the product nextc;
        }
    }
```

2.5.3 读者—写者问题

一个数据文件或记录,可被多个进程共享,把只要求读该文件的进程称为“Reader 进程”,其他进程则称为“Writer 进程”。允许多个进程同时读一个共享对象,因为读操作不会使数据文件混乱。但不允许一个 Writer 进程和其他 Reader 进程或 Writer 进程同时访问共享对象,因为这种访问将会引起混乱。所谓“读者—写者问题”是指保证一个 Writer 进程必须与其他进程互斥地访问共享对象的同步问题。

分析:(1)写者和其他写者及读者不能同时访问文件,他们之间存在互斥关系,设互斥信号量 wmutex。

(2)为了实现读者和写者之间的互斥,写者必须在读者人数为 0 时才可以访问文件,需设一个表示正在读的读者数目的变量 readcount,读者到来时应对 readcount 加 1,读完离开时应对 readcount 减 1;第一个到来的读者负责检查有无写者,最后一个离开的读者负责唤醒等待的写者。

(3)诸读者可同时读文件,但访问共享变量 readcount 应互斥,因此应设互斥信号量 rmutex。

读者—写者问题的同步算法如下:

```
semaphore wmutex=1, rmutex=1;
int readcount=0;
main()
{
    cobegin
        reader( );
        writer( );
    coend
}
reader( )
{
    while(1)
    {
        P(rmutex);
        if(readcount==0)
            P(wmutex);
        readcount=readcount+1;
        V(rmutex);
```

```
            read file;
            P(rmutex);
            readcount=readcount-1;
            if(readcount==0)
                V(wmutex);
            V(rmutex);
        }
    }
    writer( )
    {
        while(1)
        {
            P(wmutex);
            write file;
            V(wmutex);
        }
    }
```

2.6 进程通信

进程通信是指进程之间的信息交换,其所交换的信息量少则几个字节,多则成千上万个字节。进程之间的互斥和同步,由于其所交换的信息量少而被归结为低级通信。信号量机制作为同步工具是卓有成效的,但作为通信工具,其效率低且通信对用户不透明。本节所要介绍的是高级进程通信,是指用户可直接利用操作系统所提供的一组通信命令高效地传送大量数据的一种通信方式。操作系统隐藏了进程通信的实现细节,通信过程对用户是不透明的。这样就大大减少了通信程序编制上的复杂性。

2.6.1 进程通信的类型

目前,高级通信机制可归结为三大类:共享存储器系统、消息传递系统以及管道通信系统。

1.共享存储器系统

为了传输大量数据,在存储器中划出一块共享存储区,诸进程可通过对共享存储区中数据的读或写来实现通信。进程在通信前,先向系统申请获得共享存储区中的一个分区,并指定该分区的关键字。若系统已经给其他进程分配了这样的分区,则将该分区的描述符返回给申请者,然后,由申请者把获得的共享存储分区连接到本进程上。此后,便可像读、写普通存储器一样地读、写该公用存储分区。

2.消息传递系统

消息传递系统是当前应用最为广泛的一种进程间的通信机制。在该机制中,进程间的

数据交换是以格式化的消息(message)为单位的,在计算机网络中,又把 message 称为报文。程序员直接利用操作系统提供的一组通信命令(原语)实现大量数据的传递。

3. 管道通信系统

管道通信方式首创于 UNIX 系统。所谓“管道”,是指用于连接一个读进程和一个写进程以实现它们之间通信的一个共享文件,又名 pipe 文件。向管道(共享文件)提供输入的发送进程(即写进程),以字符流形式将大量的数据送入管道;而接收管道输出的接收进程(即读进程),则从管道中接收(读)数据。由于发送进程和接收进程是利用管道进行通信的,故又称为管道通信。为了协调双方的通信,管道机制必须提供以下 3 方面的协调能力:

(1)互斥,即当一个进程正在对 pipe 执行读/写操作时,其他进程必须等待。

(2)同步,指当写(输入)进程把一定数量的数据写入 pipe 时,需等待,读(输出)进程取走数据后,再将它唤醒。当读进程读空 pipe 时,也应等待,直至写进程将数据写入管道后,才将它唤醒。

(3)确定对方是否存在,只有确定了对方已存在时,才能进行通信。

2.6.2 消息传递系统通信的实现方式

在进程之间通信时,源进程可以直接或间接地将消息传送给目标进程,由此可将进程通信分为直接通信和间接通信两种通信方式。

1. 直接通信方式

这是指发送进程利用 OS 所提供的发送命令,直接把消息发送给目标进程。此时,要求发送进程和接收进程都以显式方式提供对方的标志符。通常,系统提供下述两条通信命令(原语):

send(receiver,message):发送一个消息给接收进程 receiver。

receive(sender,message):接收 sender 发来的消息。

2. 间接通信方式

间接通信方式指进程之间的通信需要通过共享数据结构——信箱,暂存发送进程发送给目标进程的消息;接收进程则从该信箱中取出对方发送给自己的消息。消息在信箱中可以安全地保存,只允许核准的目标用户随时读取。因此,利用信箱通信方式,既可实现实时通信,又可实现非实时通信。

系统为信箱通信提供了若干条原语,分别用于信箱的创建、撤销和消息的发送、接收等。

(1)信箱的创建和撤销。进程可利用信箱创建原语来创建一个新信箱。创建者进程应给出信箱名字、信箱属性(公用、私用或共享);对于共享信箱,还应给出共享者的名字。当进程不再需要读信箱时,可用信箱撤销原语将其撤销。

(2)消息的发送和接收。当进程之间要利用信箱进行通信时,必须使用共享信箱,并利用系统提供的下述通信原语进行通信:

send(mailbox,message):将一个消息发送到指定信箱。

receive(mailbox,message):从指定信箱中接收一个消息。

信箱可由操作系统创建,也可由用户进程创建,创建者是信箱的拥有者。据此,可把信箱分为以下 3 类。

(1)私用信箱。用户进程可为自己创建一个新信箱,并作为该进程的一部分。信箱的拥

有者有权从信箱中读取消息，其他用户则只能将自己构成的消息发送到该信箱中。当拥有该信箱的进程结束时，信箱也随之消失。

(2)公用信箱。它由操作系统创建，并提供给系统中的所有核准进程使用。核准进程既可把消息发送到该信箱中，也可从信箱中读取发送给自己的消息。通常，公用信箱在系统运行期间始终存在。

(3)共享信箱。它由某进程创建，在创建时或创建后指明它是可共享的，同时需指出共享进程(用户)的名字。信箱的拥有者和共享者都有权从信箱中取走发送给自己的消息。

在利用信箱通信时，在发送进程和接收进程之间存在以下4种关系：

(1)一对一关系。这时可为发送进程和接收进程建立一条两者专用的通信链路，使两者之间的交互不受其他进程的干扰。

(2)多对一关系。允许提供服务的进程与多个用户进程之间进行交互，也称为客户/服务器交互(client/server interaction)。

(3)一对多关系。允许一个发送进程与多个接收进程进行交互，使发送进程可用广播方式向接收进程发送消息。

(4)多对多关系。允许创建一个公用信箱，多个进程都能向信箱中投递消息，也可从信箱中取走属于自己的消息。

3.消息的格式

在消息传递系统中所传递的消息，必须具有一定的消息格式。通常，可把一个消息分为消息首部和消息正文两部分。消息首部包括消息在传输时所需的控制信息，如源进程名、目标进程名、消息长度、消息类型及发送的日期和时间等，而消息正文则是发送进程真正要发送的数据。

消息分为定长消息和变长消息。采用定长消息格式，可以减少对消息的处理和存储开销。这种方式可用于办公自动化系统中，为用户提供快速的便笺式通信，但这对要发送较长消息的用户是不方便的。在有的操作系统中，采用变长的消息格式，即进程所发送消息的长度是可变的，系统可能会付出更多的开销，但这方便了用户。这两种消息格式各有其优缺点，故在很多系统(包括计算机网络)中，是同时使用的。

4.进程同步方式

在进程之间进行通信时，同样需要有进程同步机制，以使诸进程间能协调通信。进程间的同步有以下3种情况：

(1)发送进程阻塞：主要用于进程之间紧密同步，即发送进程和接收进程之间无缓冲时。这两个进程平时都处于阻塞状态，直到有消息传递。

(2)发送进程不阻塞，接收进程阻塞：发送进程平时不阻塞，它可以尽快地把一个或多个消息发送给多个目标；接收进程平时处于阻塞状态，直到发送进程发来消息时才被唤醒。

(3)发送进程和接收进程均不阻塞：这也是一种较常见的进程同步形式。发送进程和接收进程平时都在忙自己的工作，仅当发生某事件使它无法继续运行时，才把自己阻塞起来等待。例如，在发送进程和接收进程之间有一个消息队列时，发送进程便可以连续地向消息队列中发送消息而不必等待；接收进程也可以连续地从消息队列中取得消息，也不必等待。

2.6.3 消息缓冲队列通信机制

消息缓冲队列通信机制首先由美国的 Hansen 提出，并在 RC4000 系统上实现，后来被广泛应用于本地进程之间的通信中。在这种通信机制中，发送进程利用 send 原语将消息直接发送给接收进程；接收进程则利用 receive 原语接收消息。

1. 数据结构

1)消息缓冲区

在消息缓冲队列通信方式中，主要利用的数据结构是消息缓冲区。它包含以下信息：

(1)sender：发送者进程标志符。

(2)size：消息长度。

(3)text：消息正文。

(4)next：指向下一个消息缓冲区的指针。

2)PCB 中有关通信的数据项

在操作系统中采用了消息缓冲队列通信机制时，除了需要为进程设置消息缓冲队列外，还应在进程的 PCB 中增加消息队列队首指针 mq，用于对消息队列进行操作，以及用于实现同步的互斥信号量 mutex 和资源信号量 sm。在 PCB 中应增加的数据项有：

(1)mq：消息队列队首指针。

(2)mutex：消息队列互斥信号量。

(3)sm：消息队列资源信号量。

2. 发送原语

发送进程在利用发送原语发送消息之前，应先在自己的内存空间设置一发送区 a，如图 2-10 所示。把待发送的消息正文、发送者进程标志符、消息长度等信息填入其中，然后调用发送原语，把消息发送给目标(接收)进程。发送原语首先根据发送区 a 中所设置的消息长度 a. size 来申请一个缓冲区 i，接着把发送区 a 中的信息复制到缓冲区 i 中。为了能将 i 挂在接收进程的消息队列 mq 上，应先获得接收进程的内部标志符 j，然后将 i 挂在 j. mq 上。由于该队列属于临界资源，故在执行 insert 操作的前后，都要执行 wait 和 signal 操作。

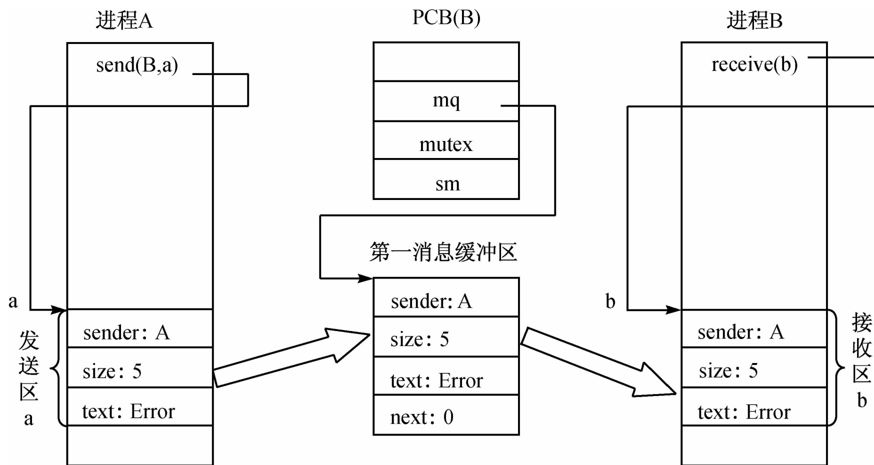

图 2-10 消息缓冲队列通信

发送原语可描述如下：

```
send(receiver,a)
{
    getbuf(a.size,i);          //根据 a.size 申请缓冲区
    i.sender=a.sender;         //将发送区 a 中的信息复制到消息缓冲区 i 中
    i.size=a.size;
    i.text=a.text;
    i.next=0;
    getid(PCBset,receiver.j);//获得接收进程内部标志符
    P(j.mutex);
    insert(j.mq,i);            //将消息缓冲区插入消息队列
    V(j.mutex);
    V(j.sm);
}
```

3.接收原语

接收进程调用接收原语 receive(b)，从自己的消息缓冲队列 mq 上取下第一个消息缓冲区 i，并将其中的数据复制到以 b 为首地址的指定消息接收区内。接收原语描述如下：

```
receive(b)
{
    j=internalname;            //j 为接收进程内部的标志符
    P(j.sm);
    P(j.mutex);
    remove(j.mq,i);            //将消息队列中第一个消息移出
    V(j.mutex);
    b.sender=i.sender;         //将消息缓冲区 i 中的信息复制到接收区 b
    b.size=i.size;
    b.text=i.text;
}
```

2.7 线 程

线程是近年来操作系统领域出现的一个非常重要的技术。线程的引入进一步提高了程序并发执行的程度，从而进一步提高了系统的吞吐量。

2.7.1 线程简介

1.线程的引入

如果说，在操作系统中引入进程的目的是为了使多个程序并发执行，以改善资源利用率及提高系统吞吐量，那么，在操作系统中再引入线程，则是为了减少程序并发执行时所付出

的时空开销,使操作系统具有更好的并发性。

进程作为资源分配的基本单位和调度的基本单位,在进行创建、撤销和进程切换时,操作系统要为其分配资源或回收资源,保存 CPU 现场信息,这些工作都需要付出较多的时间及空间开销。因此,在系统中不宜设置过多的进程,进程切换的频率也不能太高,从而限制了系统并发程度的进一步提高。

为使多个程序更好地并发执行,并尽量减少操作系统的开销,不少操作系统研究者考虑将进程的两个基本属性分离开来,分别由不同的实体来实现。为此,操作系统设计者引入了线程,让线程去完成第二个基本属性的任务(即线程是独立调度和分派的基本单位),而进程只完成第一个基本属性的任务(即进程是资源分配的基本单位)。

2. 线程的定义

线程的定义情况与进程类似,存在多种不同的提法。这些提法可以相互补充,完善对线程的理解,下面列出一些较权威的定义:

(1)线程是进程内的一个执行单元。

(2)线程是进程内的一个可调度实体。

(3)线程是执行的上下文,其含义是执行的现场数据和其他调度所需的信息(这种观点来自 Linux 系统)。

综上所述,可以将线程定义为:线程是进程内一个相对独立的、可调度的执行单元。

线程基本上不拥有系统资源,只拥有一些在运行时必不可少的资源(如程序计数器、一组寄存器和栈、线程控制块 TCB),但它可以与同属一个进程的其他线程共享进程拥有的全部资源。

多线程是指一个进程中有多个线程,这些线程共享该进程资源,这些线程驻留在相同的地址空间中,共享数据和文件。如果一个线程修改了一个数据项,其他线程可以了解和使用此结果数据。

3. 线程的状态

和进程一样,线程也有不同的状态,线程的基本状态有就绪、执行和阻塞。

(1)就绪状态。已具备执行条件,只要获得处理机就可以执行。

(2)执行状态。获得处理机正在执行。

(3)阻塞状态。因等待某事件发生而暂时无法执行。

线程间的状态转换和进程类似。

2.7.2 线程与进程的比较

由于进程与线程密切相关,因此有必要对进程与线程的异同进行比较。可以从以下几个方面对它们进行比较:

(1)拥有资源。不论是传统操作系统还是引入线程的操作系统,进程都是拥有资源的基本单位,而线程基本上不拥有系统资源,但线程可以访问其隶属进程的系统资源。

(2)调度。在引入线程的操作系统中,线程是独立调度的基本单位,进程是资源拥有的基本单位。在同一进程中,线程的切换不会引起进程切换。在不同进程中进行线程切换,如从一个进程中的线程切换到另一个进程中的线程时,将会引起进程切换。

(3)并发性。在引入线程的操作系统后,不仅进程之间可以并发执行,而且同一进程内

的多个线程之间也可以并发执行，从而使操作系统具有更好的并发性，大大提高了系统的吞吐量。

(4)系统开销。创建进程、撤销进程和切换进程时，操作系统所付出的开销远大于创建线程、撤销线程和切换线程时的开销。

2.7.3　线程的实现

在操作系统中有多种方式可实现线程：一种是在内核空间实现线程，由操作系统内核提供线程的控制机制，即内核级线程；另一种是在用户空间实现线程，由用户程序利用函数库提供线程的控制机制，即用户级线程；还有一种做法是同时在操作系统内核和用户程序两个层次上提供线程控制机制，即混合实现方式。

1.内核级线程

内核级线程是指依赖于内核，由操作系统内核完成创建、撤销和切换线程。一个内核级线程由于I/O操作而阻塞时，不会影响其他线程的运行。由于处理机时间分配的对象是线程，所以有多个线程的进程将获得更多处理机时间。

2.用户级线程

用户级线程是指不依赖于操作系统核心，由应用进程利用线程库提供创建、同步、调度和管理线程的函数来控制的线程。由于用户级线程的维护由应用进程完成，不需要操作系统内核了解用户级线程的存在，因此可以用于不支持内核级线程的多用户操作系统。用户级线程切换不需要内核特权。由于用户级线程的调度在应用进程内部进行，通常采用非抢占式和更简单的规则，也无需用户态与核心态的切换，因此速度特别快。

由于操作系统内核不了解用户级线程的存在，处理机时间分配的对象是进程，当一个线程阻塞时，整个进程都必须等待。进程内有多个线程时，每个线程的执行时间相对就少一些。

3.混合实现方式

在有些操作系统中，提供了上述两种方法的组合实现。在这种系统中，内核支持多线程的创建、调度与管理；同时，系统中又提供使用线程库的便利，允许用户应用程序建立、调度和管理用户级的线程。由于同时提供内核线程控制机制和用户线程库，因此可以很好地将内核级线程和用户级线程的优点结合起来。

本章小结

本章主要介绍了进程的基本概念、进程的状态转换、进程同步和通信以及线程的基本概念和实现。

进程是进程实体的一次运行过程，是资源分配和调度的独立单位。进程有3种基本状态：就绪、执行和阻塞，在有些系统中，还引入了挂起状态。进程的创建、撤销以及状态转换一般通过进程控制原语实现。

由于进程间存在间接制约关系和直接制约关系，系统必须协调并发进程的执行次序，即必须有进程同步机制。本章介绍了记录型信号量和P、V操作，以及如何用记录型信号量和

P、V 操作实现经典进程同步问题。进程同步问题是本章的难点。

由于进程间要交换信息，系统中提供了高级通信机制。常见的高级通信机制有共享存储器系统、消息传递系统和管道通信系统。

线程是为了减少程序并发执行时的时空开销而引入的。在引入线程的系统中，进程只是资源分配的基本单位，而线程是调度的基本单位。线程的实现有 3 种方式，一种是内核级线程，一种是用户级线程，还有些系统实现的是混合实现方式。

习　题　2

1. 什么是进程？进程有何特征？

2. 进程有哪几种基本状态？试画出进程 3 种基本状态的转换关系图。

3. 进程和线程的区别是什么？

4. 什么是临界资源？什么是临界区？对临界区的管理应遵循哪些准则？

5. 什么是线程？引入线程的原因是什么？

6. 进程间的高级通信方式有哪几类？

7. 用记录型信号量和 P、V 操作实现如图 2-11 所示的前驱关系。

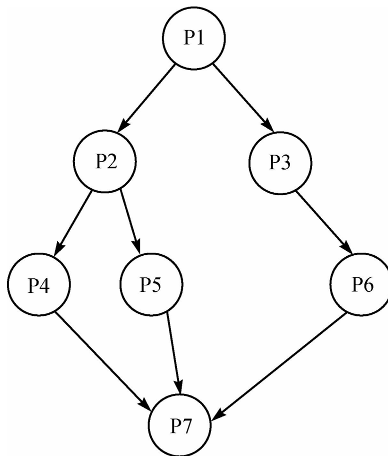

图 2-11　第 7 题图

8. 有两个用户进程 A 和 B，在执行过程中都要使用系统中同一台打印机输出各自的计算结果。

(1)试说明 A、B 两进程之间存在什么样的制约关系？

(2)为保证这两个进程能正确地打印出各自的结果，试写出利用记录型信号量机制实现进程的同步算法。

9. 设输入进程 P_I 不断向一个单缓冲区写入信息，输出进程 P_O 不断地将刚由输入进程 P_I 写入的信息从单缓冲区中读出，试问：

(1)进程 P_I、P_O 之间存在什么样的制约关系？

(2)试写出利用记录型信号量机制实现这两个进程共享缓冲区的同步算法。

10. 考虑有 3 个吸烟者进程和 1 个经销商进程的系统。每个吸烟者连续不断地做烟卷

并抽他做好的烟卷。做一支烟卷需要烟草、纸和火柴3种原料，这3个吸烟者分别掌握有烟草、纸和火柴。经销商源源不断地提供上述3种原料，但他只将其中的两种原料放在桌上，掌握第三种原料的吸烟者可做烟卷并抽烟，且在做完后为经销商发信号，然后经销商再拿出两种原料放在桌上，如此反复。试设计一个同步算法来描述他们的活动。

11. 有一阅览室，读者进入时必须先在一张登记表上登记，该表为每一座位列出一个表目，包括座号、姓名，读者离开时要注销登记信息。假如阅览室共有100个座位，试用信号量机制和P、V操作来实现用户进程的同步算法。

12. 计算进程PC和打印进程PO1、PO2共享一个单缓冲区。计算进程负责计算，并把计算结果放入单缓冲区（只能容纳一个计算结果）；打印进程PO1、PO2则负责从单缓冲区中取出计算结果进行打印，而且对每一个计算结果，PO1和PO2都需分别打印一次。请用记录型信号量实现3个进程的同步。

第3章　处理机调度与死锁

在多道程序环境下，一个作业从提交到完成，必须要经过处理机调度才能执行。在有些系统中，可能要经历多级调度，如作业调度、进程调度和对换调度。系统性能的优劣在很大程度上取决于处理机调度的性能。另外，多道程序的并发执行有可能导致死锁的发生。本章主要介绍处理机调度的类型、调度算法、死锁及其处理方法。

3.1　处理机调度的类型和准则

处理机调度分为作业调度、进程调度和对换调度，不同操作系统中所采用的调度层次不完全相同。例如，批处理系统中，通常设置作业调度和进程调度；分时系统中只需设置进程调度，一些较完善的系统中还设置了对换调度。不同系统选择调度方式和算法的准则也可能不同。

3.1.1　作业调度

作业调度的对象是外存上的作业，下面先介绍一下作业的基本概念。

1.作业

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工作的集合，包括用户程序、所需的数据及作业说明书。

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，在编写程序的过程中，通常要进行编辑、编译、链接、运行几个步骤，其中的每个步骤称为一个作业步。

在外存中往往有许多作业，为了管理调度这些作业，就必须记录进入系统中的各作业的情况，如同进程管理一样，系统为每个作业设置一个作业控制块JCB，其中记录了作业的有关信息。不同系统的JCB所包含的信息有所不同，这取决于系统对作业的要求。通常JCB中包括的主要内容有：

(1)资源要求。资源要求是指作业运行需要的资源情况，包括估计运行时间、最迟完成时间、需要的内存容量、外设类型及数量等。

(2)资源使用情况。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、内存地址和外设号等。

(3)作业的控制方式、类型和优先级等。作业的控制方式，如联机作业控制、脱机作业控制。作业类型，如终端型作业、批量型作业、I/O繁忙型作业和CPU繁忙型作业。作业的优先级是指作业进入系统运行的优先级别，优先级高的作业可以优先进入系统运行。

(4)作业名、作业状态。记录作业的标志信息及作业的当前状态。

系统在作业进入后备状态时为作业建立 JCB，当作业运行完毕进入完成状态之后，系统撤销其 JCB，释放有关资源并撤销该作业。

2. 作业的状态及转换

一个作业从进入系统到运行结束，一般需要经历提交、收容、运行、完成 4 个阶段。相应地，这 4 个阶段对应的作业处于提交、后备、运行和完成 4 种状态。作业的状态及其转换如图 3-1 所示。

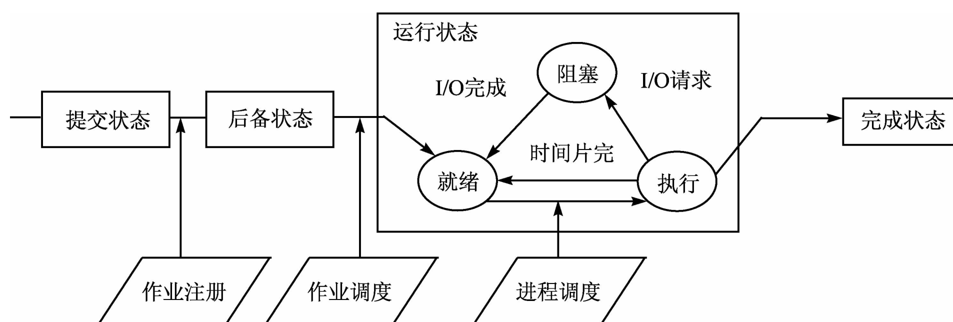

图 3-1　作业的状态及转换

1)提交状态

用户为了使用计算机解题或进行某项事务处理，必须先将作业通过输入设备提交给计算机系统。用户作业由输入设备向外存输入时所处的状态称为提交状态。

2)后备状态

当一个作业通过输入设备送入计算机，并由操作系统将其存放在磁盘输入井中以后，系统为这个作业建立一个作业控制块，并把它插入到作业后备队列中等待作业调度，此时，这个作业所处的状态称为后备状态。

3)运行状态

当作业调度程序选中一个作业，将它调入内存，为它分配必要的资源并建立了相应的进程之后，这个作业就由后备状态变为运行状态。

处于运行状态的作业在系统中并不一定真正占有处理机，它对应的进程可能处于执行状态，也可能处于就绪或阻塞状态。因此，从宏观上看，作业一旦由作业调度程序选中并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机上执行。

4)完成状态

当作业正常运行结束或因发生错误而终止运行时，作业就由运行状态转变为完成状态。此时，由操作系统将作业控制块从当前作业队列中删去，并收回其所有资源。

3. 作业调度的功能

作业调度又称宏观调度、高级调度或长程调度，是适用于批处理系统的一种调度方式。其主要功能是按照一定的原则从外存作业后备队列中选择若干个作业装入内存，给它们分配内存、I/O 设备等必要的资源，并建立相应的进程，最后将新创建的进程插入就绪队列。作业调度的运行频率较低，通常为几分钟一次。

作业调度时，每次从外存后备队列中选择多少个作业调入内存，这取决于多道程序的度。多道程序的度指的是系统中能够同时运行的作业数，应该保证作业调度后内存中的作

业数不超过多道程序的度。

在批处理系统中,新提交的作业先存放在外存上,因此需要作业调度,将它们装入内存。而在其他类型的操作系统中,通常不需要配置作业调度。

4.作业调度的时机

系统进行作业调度时,要从时间和空间两个方面考虑。首先看 CPU 是否有较多的空闲时间,其次看内存是否有足够的可用区。通常,调度一个作业的时机有以下 3 种:

(1)一个作业完成以后。当一个作业运行结束,内存中活跃的进程数量减少了。为了不至于降低处理机和内存的利用率,操作系统需要保持内存中一定量的进程。因此,有必要调度一个外存上的后备作业,使它执行。

(2)新作业提交。如果系统中的作业数量尚未使系统达到饱和的状态,处理机仍有一些闲置时间,系统在确认当前内存的道数不足的情况下,可以调度新作业,使它执行。

(3)处理机利用率低。如果内存中的进程多为 I/O 型的,它们的计算任务不足以让 CPU 忙碌起来。那么,系统可将部分等待 I/O 的进程挂起来,然后选择一个外存上的计算型作业驻留内存。

3.1.2 进程调度

进程调度的对象是内存就绪队列中的进程。

1.进程调度的功能

进程调度又称微观调度、短程调度或低级调度,其主要功能是按照某种策略从就绪队列中选取一个进程,将处理机分配给它,使它执行。进程调度的运行频率很高,一般几十毫秒就运行一次。

进程调度是操作系统中最基本的一种调度,在一般的操作系统中都必须配置进程调度。

2.进程调度方式

进程调度有两种基本的调度方式,即非抢占方式和抢占方式。

(1)非抢占方式:系统一旦将 CPU 分配给了某个进程,就使它一直执行,直到该进程完成或因某事件而阻塞时,才能将 CPU 分配给其他进程。

(2)抢占方式:系统将 CPU 分配给某进程后,可以根据某种原则终止当前进程的执行,将处理机分配给就绪队列中的其他进程。如就绪队列中一旦有优先级更高的进程出现时,系统便立刻把 CPU 分配给优先级高的新进程;或每次给进程分配一个时间片,当前进程的时间片用完时,系统立刻终止当前进程的执行,将 CPU 分配给就绪队列中新的队首进程。

非抢占方式实现简单,系统开销小,适用于大多数批处理系统,但难以满足紧急作业的需求。抢占方式可以使紧急的作业立即获得处理机,因此可用在要求严格的实时系统中,但抢占方式实现相对要复杂一些,系统开销也大。

3.进程调度的时机

引起进程调度的原因不仅与操作系统的类型有密切关系,而且还与下列因素有关:

(1)当前进程完成。

(2)当前进程由于某事件阻塞,如请求 I/O 操作、执行了 P 操作等。

(3)抢占调度方式下,一个新到来的进程比当前进程的优先级高。

(4)分配给当前进程的时间片已经用完。

3.1.3　对换调度

对换调度又称中程调度或中级调度，其主要功能是按照给定的原则和策略，将处于外存对换区中的具备执行条件的进程调入内存，或将处于内存的暂时不能执行的进程调到外存对换区。对换调度的运行频率介于作业调度与进程调度之间。

引入对换调度的主要目的是提高内存的利用率和系统吞吐量，它实际上是存储器管理中的交换功能，因此这部分内容将在存储器管理部分介绍。

3.1.4　选择调度方式和调度算法的准则

在计算机系统中，选择调度方式和调度算法受多种因素影响，如操作系统的类型、设计目标等，以下给出几种常用的准则。

1.周转时间短

周转时间的长短是批处理系统选择调度方式和调度算法的准则之一。周转时间是指从作业提交到作业完成之间的时间间隔。作业 i 的周转时间 T_i 表示如下：

$$T_i = T_{ei} - T_{si}$$

其中，T_{ei}为作业 i 的完成时间，T_{si}为作业 i 的提交时间。

从系统的角度考虑，总是希望平均周转时间短一些。平均周转时间是指多个作业周转时间的平均值。n 个作业的平均周转时间 T 可用公式表示如下：

$$T = (T_1 + T_2 + \cdots + T_n)/n$$

带权周转时间是指作业周转时间与作业实际运行时间的比。作业 i 的带权周转时间 W_i 可用公式表示如下：

$$W_i = T_i / T_{ri}$$

其中，T_i 为作业 i 的周转时间，T_{ri}为作业 i 的实际运行时间。

平均带权周转时间是指多个作业带权周转时间的平均值。n 个作业的平均带权周转时间 W 可用公式表示如下：

$$W = (W_1 + W_2 + \cdots + W_n)/n$$

2.响应时间短

响应时间的长短是分时系统选择调度方式和调度算法的准则之一。响应时间是指从用户由键盘提交一个请求开始，直至系统首次产生响应为止的一段时间。

3.保证截止时间

保证截止时间是实时系统中选择调度方式和调度算法的准则之一。截止时间是指某任务必须开始或完成的最迟时间，实时系统中如果无法保证任务的截止时间，可能造成严重的后果。

4.CPU 利用率高

CPU 是计算机系统中最重要、最昂贵的资源之一，其利用率是评价调度算法的重要指标。

5. 系统吞吐量

系统吞吐量表示单位时间内CPU完成作业的数量。对长作业来说，由于它们要消耗较长的处理机时间，因此会造成系统的吞吐量下降。而对于短作业来说，它们所需消耗的处理机时间较短，因此系统的吞吐量会提高。但调度算法和方式的不同，也会对系统的吞吐量产生较大影响。

6. 资源使用的均衡性

注意系统资源的均衡使用，使I/O繁忙型的作业与CPU繁忙型的作业搭配运行，使得CPU、内存和I/O设备都得到充分利用。

3.2 作业调度算法

作业调度是批处理系统中具有的调度，主要功能是从后备作业中按照某种算法选择若干个作业装入内存，使它们转入运行状态。本节介绍作业调度中的4种常用算法：先来先服务(FCFS)调度算法、短作业优先(SJF)调度算法、优先级(HPF)调度算法及高响应比优先(HRF)调度算法。

3.2.1 先来先服务调度算法

先来先服务调度算法是指每次从作业后备队列中选择最先进入该队列的一个或几个作业，将它们调入内存，分配必要的资源，创建进程并放入就绪队列，即哪一个作业先提交给系统，就先运行哪一个作业。

【例3-1】 设有4道作业J1、J2、J3和J4，它们的提交时间分别为10.0、10.2、10.4、10.5，要求服务时间分别为2.0、1.0、0.5、0.3(单位：小时)，试计算在单道程序环境中，采用先来先服务调度算法时每个作业的周转时间、平均周转时间和平均带权周转时间。

解：先来先服务调度算法下作业的执行情况如表3-1所示。

表3-1 先来先服务调度算法下作业的执行情况

作业名	提交时间	服务时间	开始时间	完成时间	周转时间	带权周转时间
J1	10.0	2.0	10.0	12.0	2.0	1.0
J2	10.2	1.0	12.0	13.0	2.8	2.8
J3	10.4	0.5	13.0	13.5	3.1	6.2
J4	10.5	0.3	13.5	13.8	3.3	11.0

周转时间＝完成时间－提交时间，运算结果见表3-1。

平均周转时间＝(2.0＋2.8＋3.1＋3.3)/4＝2.8

平均带权周转时间＝(1＋2.8＋6.2＋11.0)/4＝5.25

这种算法的优点是比较容易实现，对大多数用户较为公平。其缺点是不区分作业长短，对于提交较晚的短作业来说，它可能要等待很长时间才会被调度运行。此外，这种算法对一些时间要求紧迫的作业也不能做到及时处理。

3.2.2　短作业优先调度算法

短作业优先调度算法是指每次调度总是从作业后备队列中选择一个或几个估计运行时间最短的作业，将它们调入内存，分配必要的资源，创建进程并放入就绪队列。

【例 3-2】　对于【例 3-1】中的 4 道作业，试计算在单道程序环境中，采用短作业优先调度算法时每个作业的周转时间、平均周转时间和平均带权周转时间。

解：短作业优先调度算法作业的执行情况如表 3-2 所示。

表 3-2　短作业优先调度算法下作业的执行情况

作 业 名	提交时间	服务时间	开始时间	完成时间	周转时间	带权周转时间
J1	10.0	2.0	10.0	12.0	2.0	1.0
J2	10.2	1.0	12.8	13.8	3.6	3.6
J3	10.4	0.5	12.3	12.8	2.4	4.8
J4	10.5	0.3	12.0	12.3	1.8	6.0

周转时间不再详细介绍。

平均周转时间＝(2.0＋3.6＋2.4＋1.8)/4＝2.45

平均带权周转时间＝(1＋3.6＋4.8＋6.0)/4＝3.85

可以看出，由于每次选择最短的作业运行，系统的吞吐量较高，平均周转时间和平均带权周转时间较短，具有较好的调度性能。但当系统不断有较短的作业到来时，有可能使长作业一直得不到运行，致使长作业处于“饥饿”状态，这对长作业是非常不利的。这是该算法的一个明显的不足之处。另外，由于用户较难准确地估计出作业的运行时间，致使该算法难以真正做到短作业优先调度。

3.2.3　优先级调度算法

优先级调度算法也称为优先权调度算法，是基于作业运行紧迫性的一种调度算法。每个作业被赋予一个优先级，用于描述作业运行的紧迫程度。

作业调度程序每次从作业后备队列中选择一个或几个优先级最高的作业，将它们调入内存，分配必要的资源，创建进程并放入就绪队列。

这种算法总是优先调度优先级高的作业，有可能使低优先级的作业处于“饥饿”状态。

3.2.4　高响应比优先调度算法

高响应比优先调度算法是 FCFS 和 SJF 两种算法的折中，既考虑作业进入系统的时间，又顾及作业的运行时间的长度。响应比 R 可以定义如下：

$$\text{响应比 } R=\frac{\text{等待时间}+\text{服务时间}}{\text{服务时间}}=\frac{\text{等待时间}}{\text{服务时间}}+1$$

作业等待时间是指一个作业从进入系统直到本次调度的时间。作业服务时间即作业的估计运行时间。

由响应比的计算公式可以看出：

(1)作业服务时间越短响应比越高,短小作业越能较快得到系统响应。因此该算法有利于短作业。

(2)对于服务时间相同的作业,等待时间长的作业响应比高,优先得到调度,即先来的作业优先。

(3)一个服务时间很长的作业在系统中等待的时间不断增加,响应比就会不断提高,被调用的可能性随之不断增大。这种算法可以避免大作业在系统中长期得不到调度。

因此,该算法既照顾了短作业,又兼顾了长作业,不足之处是每次调度都要计算所有后备作业的响应比,增加了系统开销。

3.3 进程调度算法

进程调度是任何类型操作系统中都具备的调度,也是最基本的调度,其性能优劣直接影响操作系统的性能。根据不同的系统设计目标,可选择不同的进程调度算法。常见的进程调度算法有先来先服务调度算法、短进程优先调度算法、优先级调度算法、时间片轮转调度算法和多级反馈队列调度算法。

3.3.1 先来先服务调度算法

先来先服务(FCFS)调度算法是指每次从就绪队列中选择最先进入该队列的进程,将处理机分配给它,使之执行,该进程一直执行下去,直到完成或因某种原因而阻塞时才释放处理机。

同先来先服务作业调度算法一样,该算法对长进程有利,对短进程不利。另外,该算法有利于 CPU 繁忙型作业,不利于 I/O 繁忙型作业。CPU 繁忙型作业指需要大量的 CPU 时间进行计算,而很少请求 I/O 的作业;I/O 繁忙型作业指需要频繁请求 I/O 的作业。

3.3.2 短进程优先调度算法

短进程优先(SPF)调度算法是指每次从就绪队列中选择估计执行时间最短的进程,将处理机分配给它,使之执行,该进程一直执行下去,直到完成或因某种原因而阻塞时才释放处理机。

该算法可能使长进程长期得不到调度。

3.3.3 优先级调度算法

优先级(HPF)调度算法是指每次把 CPU 分配给就绪队列中具有最高优先级的就绪进程,使它执行。在不同的调度方式下,调度时机也不同。

1. 调度方式

根据已占有 CPU 的进程是否可被抢占,可把优先级调度算法分为非抢占式优先级调度算法和抢占式优先级调度算法。

(1)非抢占式优先级调度算法是指每次将 CPU 分配给就绪队列中具有最高优先级的就绪进程,使它执行,直至该进程完成或因某事件而阻塞时,才将 CPU 重新分配给其他优先级最高的进程。

(2)抢占式优先级调度算法是指每次将CPU分配给就绪队列中具有最高优先级的就绪进程,使它执行,就绪队列中一旦有优先级更高的进程出现时,系统便立刻把CPU分配给优先级更高的新进程。

【例3-3】 系统中有4个进程A、B、C、D,它们到达内存就绪队列的时间分别是0、0、0、5,要求的服务时间分别是10、1、5、3,优先级分别为3、5、2、4,请分别计算采用非抢占式优先级调度算法和抢占式优先级调度算法时各进程的周转时间和平均周转时间。

解:(1)非抢占式优先级调度算法下进程的执行情况如图3-2所示。

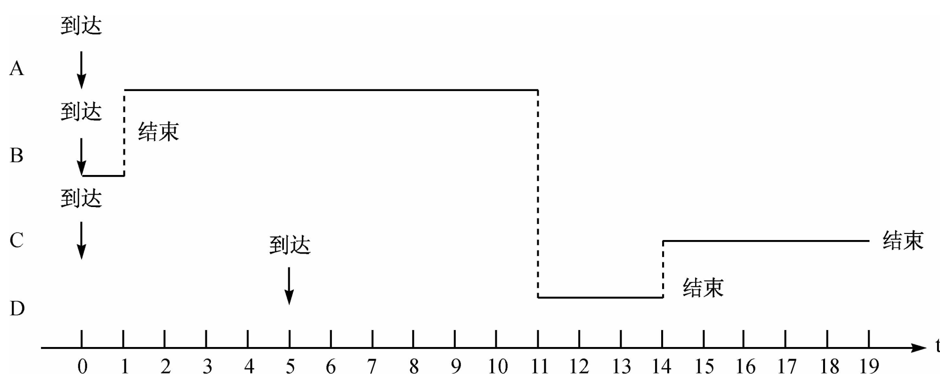

图3-2　非抢占式优先级调度算法下进程的执行情况

进程周转时间分别为:A:11,B:1,C:19,D:9

平均周转时间=(11+1+19+9)/4=10

(2)抢占式优先级调度算法下进程的执行情况如图3-3所示。

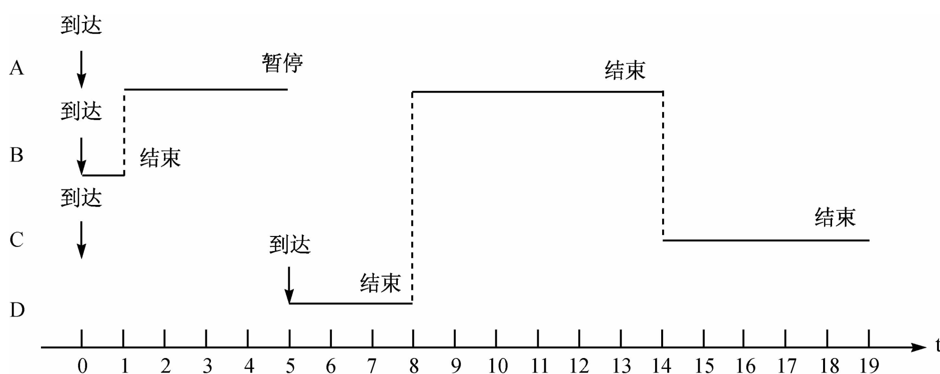

图3-3　抢占式优先级调度算法下进程的执行情况

进程周转时间分别为:A:14,B:1,C:19,D:3

平均周转时间=(14+1+19+3)/4=9.25

非抢占式优先级调度算法和抢占式优先级调度算法的区别在于进程调度的时机不同。非抢占式优先级调度算法实现起来简单,但不能满足紧急进程的要求;抢占式优先级调度算法能优先处理“紧迫”进程,但实现复杂,系统开销大。

2. 优先级的类型

通常给进程赋予某范围(如0～255)的一个整数来表示进程优先级的高低,该数字称为

进程的优先数。在有些系统中,优先数越大,优先级越高,有些系统则相反。

进程优先级在进程创建时确定,根据优先级是否发生改变将优先级分为静态优先级和动态优先级。

1)静态优先级

静态优先级在进程创建时确定,在进程整个生命周期内保持不变。静态优先级通常根据以下因素确定:

(1)进程的类型。通常系统进程的优先级高于用户进程的优先级,前台用户进程的优先级高于后台用户进程的优先级。

(2)进程所需的资源情况。根据进程的资源需求量来确定优先级,如估计运行时间、所需内存大小、I/O 的数量等。通常规定进程优先级与进程所需资源量多少成反比,即资源请求越多的进程优先级越低,反之越高。

(3)作业的优先级。根据作业的优先级来决定其所属进程的优先级。例如,一种常用于多道批处理系统的方法是系统把用户作业说明书上提供的外部优先级赋给该作业及其所创建的进程。

静态优先级实现简单,系统开销小。但当系统中不断有高优先级的进程到来时,低优先级的进程可能处于“饥饿”状态。为此,又引入了动态优先级。

2)动态优先级

动态优先级在进程创建时确定,随着进程推进优先级发生变化。

(1)随着就绪进程等待 CPU 时间的增加,优先级不断提高。一个初始时优先级较低的进程,在就绪队列中等待的时间越长,其优先级变得越高,一段时间以后其优先级将变得最高而获得 CPU。

(2)随着进程占有 CPU 时间的增加,优先级不断降低。在采用抢占调度方式的系统中,规定进程优先级随着执行时间的增加而降低,可防止一个进程长时间垄断 CPU。

3.3.4　时间片轮转调度算法

时间片轮转(RR)调度算法主要用于分时系统中的进程调度。在时间片轮转调度算法中,系统将所有就绪进程按到达时间的先后次序排成一个队列,进程调度程序总是选择就绪队列中的队首进程,让它执行一个固定的时间片(如 50 ms),时间片结束时若该进程未完成,系统便将它送至就绪队列末尾,再把处理机分配给就绪队列的队首进程。这样,就绪队列中的进程轮流执行一个时间片,如此反复,直到完成为止。

在时间片轮转调度算法中,时间片的大小对系统性能的影响很大。如果时间片足够大,以致所有进程都能在一个时间片内执行完毕,该算法就退化成先来先服务调度算法,进程的响应时间将很长。如果时间片很小,则进程响应时间短,但处理机将在进程之间频繁切换,系统开销较大。因此时间片的选择应大小适当。

【例 3-4】　某分时系统中有 A、B、C、D 四个进程,到达时间分别为 0、1、2、3,要求服务时间分别为 6、2、4、3,采用时间片轮转调度算法,时间片大小为 1,请给出这 4 个进程的周转时间和平均周转时间。

解: 这 4 个进程在系统中的运行情况如图 3-4 所示。

各进程的周转时间分别为:

A:15,B:4,C:12,D:9

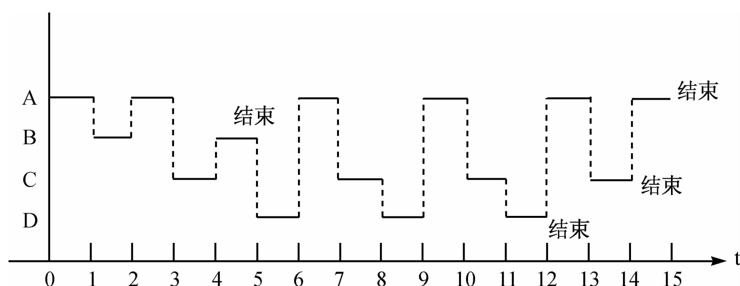

图 3-4　时间片轮转调度算法下进程的执行情况

平均周转时间＝(15＋4＋12＋9)/4＝10

3.3.5　多级反馈队列调度算法

多级反馈队列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，多级反馈队列调度算法可以兼顾多方面的系统目标。

多级反馈队列调度算法的实现思想如下：

(1)就绪队列和时间片大小的设置。设置多个就绪队列，各队列赋予不同的优先级。第一个队列优先级最高，第二个队列次之，逐个降低；队列优先级越高，时间片越小，反之，时间片越大。

(2)进程排队原则。新进程进入内存，首先进入第一个队列的末尾，按时间片原则接受调度。若该进程在一个时间片结束时未完成，则该进程进入下一队列末尾，同样按时间片原则接受调度，……，若进程一直降到第 n 个(最后一个)队列，便停留在此队列中按时间片原则接受调度。

(3)调度。每次调度选择当前优先级最高的非空就绪队列中的进程；若正在执行第 i 个队列中的某进程时，有进程进入某高优先级的队列，则该进程抢占处理机，将当前进程插入第 i 个队列的末尾。

多级反馈队列调度算法具有较好的性能。对终端型作业而言，由于这类作业需要的处理机时间较短，因而能够在前一两个队列中完成，从而保证了终端型作业具有较快的响应时间；同样，短作业能在前几个队列中完成，因而其周转时间较短；而长作业可以依次在各队列中得到服务。这种方法既照顾了时间紧迫的进程，又兼顾了短进程同时考虑了长进程，是一种比较理想的进程调度方法。

3.4　死锁的基本概念

在多道程序系统中，由于多个进程并发执行，改善了系统资源的利用率并提高了系统的吞吐量，但可能导致死锁。例如，有一座独木桥，如果有两人分别从桥的两边上桥，当他们在桥上相遇时，若他们互不退让，就会出现谁也不能过河的局面。

死锁是指多个并发执行的进程因竞争系统资源而造成的一种僵局，若无外力作用，这些进程都将无法向前推进。

3.4.1 产生死锁的原因

死锁产生的主要原因有两方面：一方面是进程竞争资源，即由于系统资源不足，在多个进程竞争资源的过程中可能会导致死锁的产生；另一方面，进程间推进顺序不当也会引起死锁，即在多个进程的执行过程中，请求和释放资源的顺序不当也会导致死锁。

1.竞争资源引起死锁

系统中的资源分为可剥夺性资源和不可剥夺性资源。相应的，竞争资源有两种。

1)竞争可剥夺性资源

可剥夺性资源是指进程获得这种资源后，该资源又可被系统或其他进程剥夺。由于CPU可以被高优先级的进程抢占，因此，CPU属于可剥夺性资源；由于作业可以在内存中移动，即进程的内存可被系统剥夺，因此，内存也属于可剥夺性资源。进程在竞争这类资源时不会引起死锁。

2)竞争不可剥夺性资源

不可剥夺性资源是指进程获得这种资源后，除非进程用完后自行释放，系统不能强行收回。如磁带机、打印机等大多数物理设备和一些共享数据结构都属于不可剥夺性资源。竞争不可剥夺性资源可能引起死锁。例如，某计算机只有一台打印机和一台输入设备，进程P1正占用输入设备，同时又提出使用打印机的请求，但此时打印机正被进程P2所占用，而P2在未释放打印机之前，又提出请求使用正在被P1占用的输入设备。这样两个进程相互无休止地等待下去，均无法继续执行，此时两个进程陷入死锁状态，如图3-5所示。

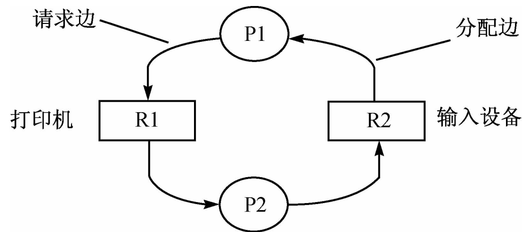

图3-5 竞争不可剥夺性资源的死锁情况

2.进程间推进顺序不当引起死锁

竞争资源可能引起死锁，但不是必然，只有在竞争资源过程中，请求和释放资源的顺序不当时才会产生死锁。如图3-5所示的例子中，若按照以下顺序执行：

P1：request(R2)

P1：request(R1)

P2：request(R1)

P1：release(R2)

P1：release(R1)

P2：request(R2)

P2：release(R1)

P2：release(R2)

则P1、P2都可以顺利完成，如图3-6所示曲线①。但若按照下面的顺序：

P1：request(R2)
P2：request(R1)
P1：request(R1)
P2：request(R2)
P1：release(R2)
P1：release(R1)
P2：release(R1)
P2：release(R2)

执行语句 P2：request(R1)后，P1 进程保持了资源 R2，P2 保持了资源 R1，继续执行将进入不安全区 D。执行到 P2：request(R2)时，P1、P2 两进程都将阻塞，无法向前推进，导致死锁发生，如图 3-6 所示曲线②。

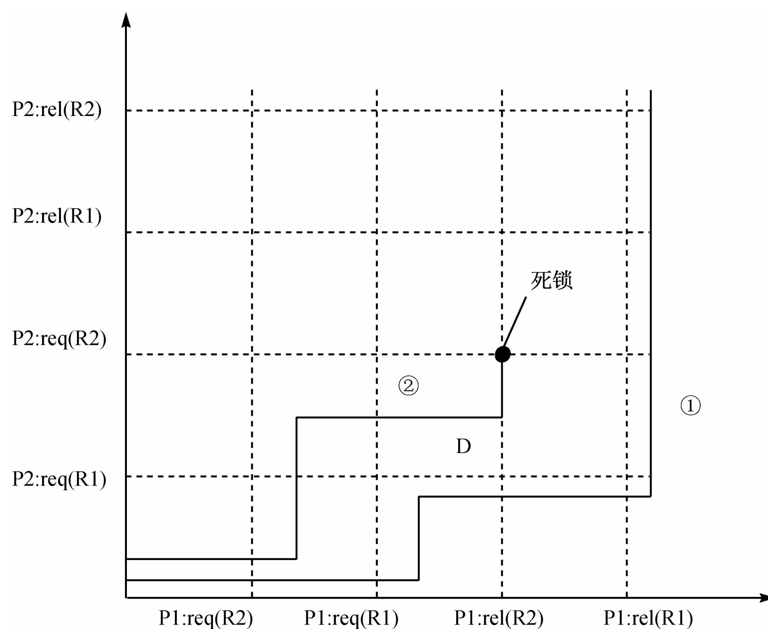

图 3-6　进程间推进顺序不当引起死锁

3.4.2　产生死锁的必要条件

死锁的发生必须具备一定的条件。死锁的产生必须同时具备以下 4 个必要条件：

(1)互斥条件。进程要求对所分配的资源进行排他性控制，即在一段时间内某资源仅为一个进程所占有。

(2)不剥夺条件。进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走，即只能由获得该资源的进程自行释放。

(3)请求和保持条件。在等待分配新资源的同时，进程继续占有已分配到的资源。请求和保持条件又称为部分分配条件。

(4)循环等待条件。存在一种进程资源循环等待链，链中的每一个进程已获得的资源同时被链中的下一个进程所请求。

3.4.3　处理死锁的基本方法

目前用于处理死锁的方法主要有以下几种：

(1)预防死锁。通过设置某些限制条件，去破坏产生死锁的4个必要条件中的一个或几个来防止发生死锁。

(2)避免死锁。在资源的动态分配过程中，用某种方法防止系统进入不安全状态，从而避免死锁。

(3)检测及解除死锁。通过系统的检测机构及时地检测出死锁，然后采取某种措施解除死锁。

3.5　死锁的预防和避免

死锁的预防和避免都是通过施加某些限制条件，来防止死锁的发生。

3.5.1　死锁的预防

预防死锁是一种静态的解决死锁问题的方法。为了使系统安全运行，应在设计操作系统时，对资源的使用进行适当限制，预防系统在运行过程中产生死锁。由于产生死锁的4个必要条件必须同时存在，系统才会产生死锁，所以，只要使4个必要条件中至少有一个不能成立，就可以达到预防死锁的目的。由于互斥性是某些资源的固有特性(如打印机是一种不能同时供多个进程使用的互斥资源)，所以一般不破坏互斥条件。

1)破坏不剥夺条件

让进程逐个提出对资源的请求。当一个已经保持了某些资源的进程，在提出新的资源请求而不能立即得到满足时，必须释放它已经保持了的所有资源，待以后需要时再重新申请。这就意味着某一进程已经占有的资源，在运行过程中会被暂时地释放掉，也可以认为是被剥夺了，从而摒弃了不剥夺条件。

这种方法实现起来比较复杂且要付出很大代价。因为一个资源在使用一段时间后，资源被迫释放可能造成前段工作的失效，例如，进程在运行的过程中，已用打印机输出一部分信息但中途又因申请另一资源未果而被迫暂停运行并释放打印机，后来系统又把打印机分配给其他进程使用。当进程再次恢复运行并再次获得打印机继续打印时，会使得前后两次打印输出的数据不连续，即打印输出的信息，中间有一段是另一进程的。此外，这种策略还可能因为反复地申请和释放资源，致使进程的执行被无限地推迟，不仅延长了进程的周转时间，而且增加了系统开销，降低了系统吞吐量。

2)破坏请求和保持条件

采用静态分配策略为进程分配资源。在进程执行之前，若系统能满足某进程的全部资源请求，就将该进程所需的全部资源分配给它，这样，在进程执行过程中不再请求新的资源，就不会有死锁出现。否则，如果当时的系统资源不能一次满足它的要求，则该进程不能执行。

这种方法对于那些“紧俏”的资源来说，进程在整个生命期中一直占用它们，造成了资源

的极大浪费。另外,仅当进程获得了所需的全部资源后方能运行,将造成进程执行时间延迟。

3)破坏循环等待条件

采用资源有序分配法来破坏循环等待条件。为每类资源编排序号,规定进程只能按资源号递增的顺序申请资源。采用这种方法,系统在任何情况下都不可能进入循环等待的状态。资源编序这种做法在于如何给资源确定各方面都比较满意的序号,另外,这种方法限制了新类型设备的增加。

3.5.2 死锁的避免

预防死锁所采取的几种策略对资源的使用施加了较强的限制条件,损害了系统性能。在死锁的避免方法中,不对进程申请资源施加限制条件,而是检查进程的资源申请是否会导致系统进入不安全状态,只要能使系统始终处于安全状态,便可以避免死锁的发生。

1.安全状态

如果在某时刻,系统能按某种进程顺序如＜P1,P2,…,Pn＞来为每个进程分配其所需的资源,直至最大需求,使每个进程都可以顺利执行完成,则称此时的系统处于安全状态,称序列＜P1,P2,…,Pn＞为安全序列。若某一时刻系统中不存在任何一个安全序列,则称此时的系统状态为不安全状态。

并非所有的不安全状态都必然转化为死锁状态,但当系统进入不安全状态后,便可能进入死锁状态;反之,只要系统处于安全状态,便可以避免进入死锁状态。因此,在避免死锁的方法中,允许进程动态地申请资源,系统在进行资源分配之前,先计算资源分配的安全性。若此次分配不会导致系统进入不安全状态,便将资源分配给进程,否则进程等待。

2.安全状态向不安全状态的转换

假定某系统中有 R1、R2 和 R3 三类资源,在 T0 时刻 P1、P2、P3 和 P4 这 4 个进程对资源的占用和需求情况见表 3-3,此时系统的剩余资源数量为(2,1,2)。

表 3-3 T0 时刻 4 个进程对资源的占用和需求情况

<table>
<tr><th rowspan="2">进 程</th><th colspan="3">最大资源需求量</th><th colspan="3">已分配资源数量</th><th colspan="3">剩余资源数量</th></tr>
<tr><th>R1</th><th>R2</th><th>R3</th><th>R1</th><th>R2</th><th>R3</th><th>R1</th><th>R2</th><th>R3</th></tr>
<tr><td>P1</td><td>3</td><td>2</td><td>2</td><td>1</td><td>0</td><td>0</td><td rowspan="4">2</td><td rowspan="4">1</td><td rowspan="4">2</td></tr>
<tr><td>P2</td><td>6</td><td>1</td><td>3</td><td>4</td><td>1</td><td>1</td></tr>
<tr><td>P3</td><td>3</td><td>1</td><td>4</td><td>2</td><td>1</td><td>1</td></tr>
<tr><td>P4</td><td>4</td><td>2</td><td>2</td><td>0</td><td>0</td><td>2</td></tr>
</table>

T0 时刻,剩余资源数量可满足 P2 的需求,使 P2 达到最大资源需求量,从而顺利完成并释放资源,系统剩余资源数量将变为(6,2,3),又可以满足 P1 的需求,P1 运行完以后系统资源又可以满足 P3、P4 的需求,存在安全序列＜P2,P1,P3,P4＞,因此 T0 时刻系统处于安全状态。

T0 时刻进程 P1 请求 1 个 R1 资源和 1 个 R3 资源,若分配给进程 P1,则各进程对资源的占用和需求情况见表 3-4。

表 3-4 满足 P1 的请求后各进程对资源的占用和需求情况

进程	最大资源需求量 R1	R2	R3	已分配资源数量 R1	R2	R3	剩余资源数量 R1	R2	R3
P1	3	2	2	2	0	1	1	1	1
P2	6	1	3	4	1	1			
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

此时剩余资源数量为(1,1,1),已不能满足任何进程的需求,系统将进入不安全状态。因此,在给 P1 分配资源之前应该先检查系统是否会进入不安全状态,若是,则暂时先不分配。

3.5.3 银行家算法

避免死锁的著名算法是 Dijkstra 在 1965 年提出的银行家算法。由于此算法类似于银行系统现金借款的发放,故称为银行家算法。

1. 算法中的数据结构

(1)可利用资源向量 Available。这是一个含有 m 个元素的一维数组,其中的每一个元素代表一类资源的空闲资源数目,其初始值是系统中所配置的该类资源数目,其数值随该类资源的分配和回收而动态地改变。如果 Available[j]=k,表示系统中现有空闲的 Rj 类资源 k 个。

(2)最大需求矩阵 Max。这是一个 n×m 的矩阵,它定义了系统中每一个进程对资源的最大需求数目。如果 Max[i,j]=k,表示进程 Pi 需要 Rj 类资源的最大数目为 k。

(3)分配矩阵 Allocation。这是一个 n×m 的矩阵,它定义了系统中当前已分配给每一个进程的各类资源数目。如果 Allocation[i,j]=k,表示进程 Pi 当前已分到 Rj 类资源的数目为 k。

$Allocation_i$ 表示进程 Pi 的分配向量,由矩阵 Allocation 的第 i 行构成。

(4)需求矩阵 Need。这是一个 n×m 的矩阵,它定义了系统中每一个进程还需要各类资源的数目。如果 Need[i,j]=k,表示进程 Pi 还需要 Rj 类资源 k 个,才能完成其任务。

$Need_i$ 表示进程 Pi 的需求向量,由矩阵 Need 的第 i 行构成。

上述数据结构存在关系:Need[i,j]=Max[i,j]-Allocation[i,j]

2. 算法思想

设 $Request_i$ 是进程 Pi 的请求向量,$Request_i$[j]=k 表示进程 Pi 请求分配 Rj 类资源 k 个。当 Pi 发出资源请求后,系统进行下述检查:

(1)如果 $Request_i \leqslant Need_i$,则转向步骤(2);否则出错,因为进程所需要的资源数目已超出它所宣布的最大值。

(2)如果 $Request_i \leqslant Available$,则转向步骤(3);否则,表示系统中尚无足够的资源满足进程 Pi 的申请,Pi 必须等待。

(3)系统试着把请求的资源分配给进程 Pi,并修改下面数据结构中的数值:

$Available=Available-Request_i$;

$Allocation_i$ = $Allocation_i$ + $Request_i$;

$Need_i$ = $Need_i$ − $Request_i$;

(4)系统执行安全性算法，检查此次资源分配后，系统是否处于安全状态。若安全，才正式将资源分配给进程 Pi，以完成此次分配；否则，将试探分配作废，恢复原来的资源状态，让进程 Pi 等待。

3. 安全性算法

系统所执行的安全性算法描述如下：

(1)设置两个向量 Work 和 Finish：Work 表示系统可提供给进程继续运行的各类资源的空闲资源数目，它含有 m 个元素，执行安全性算法开始时，Work＝Available。Finish 表示是否有足够的资源分配给进程，使之运行完成。开始时，Finish[i]＝false；当有足够资源分配给进程 Pi 时，令 Finish[i]＝true；i＝1，2，…，n。

(2)从进程集合中找到一个 Finish[i]＝false 且 $Need_i \leqslant Work$ 的进程，如找到则执行步骤(3)；否则执行步骤(4)。

(3)当进程 Pi 获得资源后，可顺利执行直到完成，并释放出分配给它的资源，故应执行：

```
Work＝Work＋Allocation;
Finish[i]＝true;
```

然后转向第(2)步。

(4)若所有进程 Finish[i]都为 true，则表示系统处于安全状态；否则，系统处于不安全状态。

4. 银行家算法示例

假定系统中有 4 个进程 P1、P2、P3 和 P4，三类资源 R1、R2 和 R3，数量分别为 9、3、6。在 T0 时刻的资源分配情况见表 3-5，试问：

(1)T0 时刻是否安全？

(2)T0 时刻以后，若进程 P2 发出资源请求 $Request_2$(1，0，1)，系统能否将资源分配给它？

(3)在进程 P2 申请资源后，若 P1 发出资源请求 $Request_1$(1，0，1)，系统能否将资源分配给它？

(4)在进程 P2 申请资源后，若 P3 发出资源请求 $Request_3$(0，0，1)，系统能否将资源分配给它？

表 3-5　T0 时刻的资源分配表

进程	Max			Allocation			Need			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	2	2	2	1	1	2
P2	6	1	3	5	1	1	1	0	2			
P3	3	1	4	2	1	1	1	0	3			
P4	4	2	2	0	0	2	4	2	0			

解：(1)利用安全算法对 T0 时刻的资源分配情况进行检查，可得到如表 3-6 所示的 T0 时刻的安全性检查，从中得知，T0 时刻存在着一个安全序列<P2，P1，P3，P4>，因此系统是安全的。

表 3-6 T0 时刻的安全性检查

进程	Work			Need			Allocation			Work+Allocation			Finish
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	
P2	1	1	2	1	0	2	5	1	1	6	2	3	true
P1	6	2	3	2	2	2	1	0	0	7	2	3	true
P3	7	2	3	1	0	3	2	1	1	9	3	4	true
P4	9	3	4	4	2	0	0	0	2	9	3	6	true

(2)P2 发出请求向量 Request$_2$(1,0,1),系统按银行家算法进行检查:

①Request$_2$(1,0,1)$\leqslant$Need$_2$(1,0,2)

②Request$_2$(1,0,1)$\leqslant$Available(1,1,2)

③试探为 P2 分配资源,并修改 Available、Allocation$_2$、Need$_2$ 向量,由此形成的资源变化情况见表 3-7。

表 3-7 P2 申请资源后的资源分配表

进程	Max			Allocation			Need			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	2	2	2	0	1	1
P2	6	1	3	6	1	2	0	0	1			
P3	3	1	4	2	1	1	1	0	3			
P4	4	2	2	0	0	2	4	2	0			

④再利用安全算法检查此时系统是否安全,可得表 3-8 所示的安全性检查。

表 3-8 P2 申请资源后的安全性检查

进程	Work			Need			Allocation			Work+Allocation			Finish
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	
P2	0	1	1	0	0	1	6	1	2	6	2	3	true
P1	6	2	3	2	2	2	1	0	0	7	2	3	true
P3	7	2	3	1	0	3	2	1	1	9	3	4	true
P4	9	3	4	4	2	0	0	0	2	9	3	6	true

经安全性检查得知,可以找到一个安全序列<P2,P1,P3,P4>。因此,系统是安全的,可以立即将 P2 所申请的资源分配给它。

(3)P1 发出请求向量 Request$_1$(1,0,1),系统按银行家算法进行检查:

①Request$_1$(1,0,1)$\leqslant$Need$_1$(2,2,2)

②Request$_1$(1,0,1)$>$Available(0,1,1),让 P1 等待。

(4)P3 发出请求向量 Request$_3$(0,0,1),系统按银行家算法进行检查:

①Request$_3$(0,0,1)$\leqslant$Need$_3$(1,0,3)

②Request$_3$(0,0,1)$\leqslant$Available(0,1,1)

③系统试探为 P3 分配资源,并修改有关数据,见表 3-9。

表 3-9　P3 申请资源后的资源分配表

进　程	Max			Allocation			Need			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	2	2	2	0	1	0
P2	6	1	3	6	1	2	0	0	1			
P3	3	1	4	2	1	2	1	0	2			
P4	4	2	2	0	0	2	4	2	0			

④用安全性算法检查系统是否安全。从表 3-9 中可以看出,可用资源 Available(0,1,0)已不满足任何进程的需要,故系统进入不安全状态,此时系统不能分配资源。

3.6　死锁的检测与解除

死锁预防和死锁避免算法都是在系统为进程分配资源前或分配资源时施加限制条件或进行检测。如果在分配资源时系统没有采取任何措施,就必须提供死锁检测和解除的手段。

死锁的检测和解除则是为进程分配资源时不采取任何措施,而是通过系统所设置的检测机构,及时地检测出死锁的发生,然后采取适当的措施,将死锁解除掉。因此,系统中必须保存资源的请求和分配信息,并提供一种算法来检测系统是否已进入死锁状态。

3.6.1　死锁的检测

检测死锁的基本思想是在操作系统中保存资源的请求和分配信息,利用某种算法对这些信息加以检查,以判断是否存在死锁。为此,系统中用资源分配图来描述进程和资源间的申请和分配关系。

1.资源分配图

资源分配图又称进程—资源图。该图由一组结点和一组边构成,结点分为进程结点和资源结点,进程结点用圆圈表示,资源结点用方框表示,方框里的小圆圈的个数表示这类资源的数目。图中的有向边也分两种,一种是由进程指向资源的有向边,称为请求边,表示该进程请求一个某类资源;一种是由资源指向进程的有向边,称为分配边,表示已分配给该进程一个某类资源。

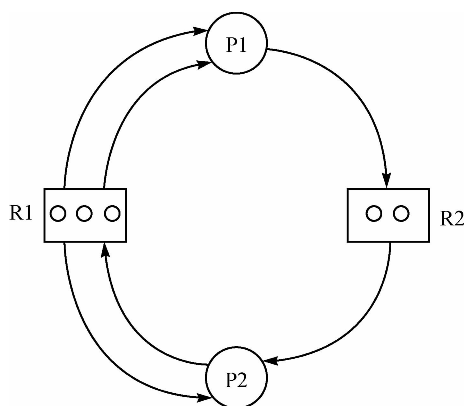

图 3-7　资源分配图

如图 3-7 所示的资源分配图中,有三个 R1 资源和两个 R2 资源,已分配给进程 P1 两个 R1 资源,分配给进程 P2 一个 R1 资源和一个 R2 资源。进程 P1 请求一个 R2 资源,进程 P2 请求一个 R1 资源。

2. 死锁定理

可以通过将资源分配图简化的方法来检测系统状态 S 是否是死锁状态。简化方式如下：

(1)在资源分配图中，找出一个既不孤立又不阻塞的进程结点 Pi(即从进程集合中找到一个有边相连，且资源申请数量小于或等于系统中已有空闲资源数量的进程结点)。进程 Pi 获得了它所需的全部资源，它就能继续运行直至完成，然后释放它所占有的所有资源。这相当于消去 Pi 的所有请求边和分配边，使之成为孤立的结点。

(2)重复执行步骤(1)，直至找不出一个既不孤立又不阻塞的进程结点为止。

若能消去图中所有的边，使所有进程结点成为孤立结点，则称该图是可完全简化的，否则称该图是不可完全简化的。

死锁定理：S 为死锁状态的条件是 S 状态的资源分配图不可完全简化。

在图 3-7 中，进程 P1 是一个既不孤立又不阻塞的进程结点，将 P1 的两条分配边和一条请求边消去，便形成了如图 3-8(a)所示的情况。进程 P1 释放资源后，可以唤醒因等待这些资源阻塞的进程，原来阻塞进程可能变为非阻塞进程。图 3-7 的进程 P2 原为阻塞进程，但在图 3-8(a)中变成了非阻塞进程。重复执行步骤(1)，消去 P2 的两条分配边和一条请求边，就形成了如图 3-8(b)所示的情况。

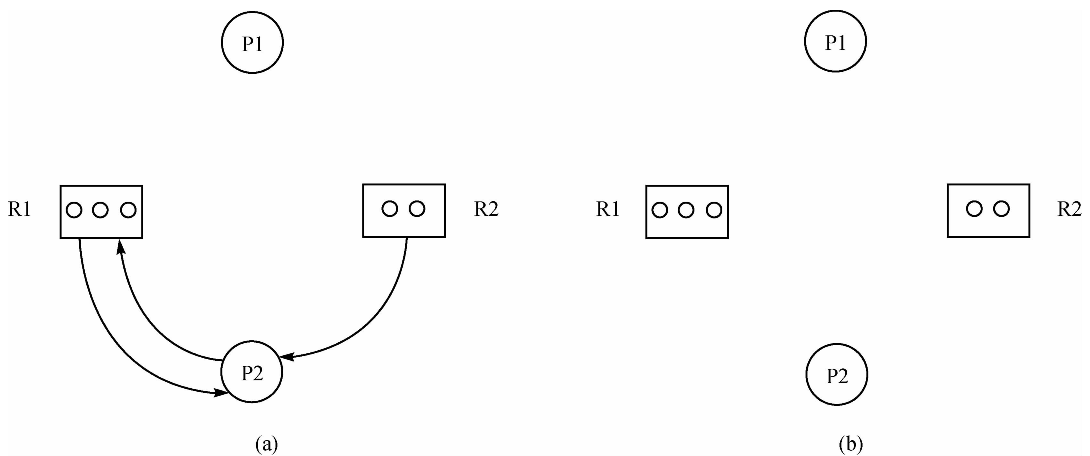

图 3-8　资源分配图的简化

3. 死锁的检测算法

死锁检测的思想是考查某一时刻系统状态是否合理，即是否存在一组可以实现的系统状态，能使所有进程都得到它们所申请的资源而运行结束，其实现算法思想如下：获得某时刻 t 系统中各类可利用资源的数目 w(t)，对于系统中的一组进程<P1，P2，…，Pn>，找出那些对各类资源的需求数量均小于等于系统现有的各类资源可用数目的进程，这样的进程可以获得它们所需要的全部资源并运行结束，当它们运行结束后将释放所占有的全部资源，从而使可用资源数目增加，将这样的进程加入到可运行结束的进程序列 L 中(检测开始时，L 为空)，然后对剩下的进程再作上述考查。如果一组进程<P1，P2，…，Pn>中有一个或几个进程不属于序列 L，那么它们将陷入死锁状态。

3.6.2　死锁的解除

用以上方法一旦检测出系统已经出现了死锁，就应将陷入死锁的进程从死锁状态中解脱出来，常用解除死锁的方法有两种：

(1)资源剥夺法。当发现死锁后，从其他进程那里剥夺足够数量的资源给死锁进程，以解除死锁状态。

(2)撤销进程法。最简单的方法是撤销全部死锁进程，使系统恢复到正常状态，但这种做法付出的代价太大。另一种方法是按照某种顺序逐个撤销死锁进程，直到有足够的资源供其他未被撤销的进程使用，直到消除死锁状态为止。可以按照付出代价最小的顺序撤销进程，也可以按照使撤销进程数最少的顺序撤销进程。

本章小结

本章主要介绍了处理机调度的基本概念，常见的作业和进程调度算法，死锁的基本概念及解除死锁的方法。

处理机调度分为3级：作业调度、进程调度和对换调度。其中，作业调度是批处理系统中采用的调度，进程调度是任何类型的操作系统中都必须具备的调度，是最基本的调度。在选择调度算法时，不同的系统设计目标有不同的准则。常见的作业调度算法有先来先服务(FCFS)、短作业优先(SJF)、优先级(HPF)和高响应比优先(HRF)调度算法，进程调度算法有先来先服务(FCFS)、短进程优先(SPF)、优先级(HPF)和时间片轮转(RR)调度算法。

多个进程并发执行将提高系统资源利用率和系统的吞吐量，但也有可能产生死锁，可以通过死锁预防、死锁避免和死锁检测及解除的方法来处理死锁。

通过本章的学习，读者应该掌握处理机调度的基本概念，能够给出不同调度算法下作业或进程的执行情况，计算进程的周转时间和带权周转时间。熟练掌握死锁的基本概念，掌握处理死锁的方法，熟练掌握银行家算法。

习　题　3

1. 作业调度的主要功能是什么？进程调度的功能是什么？

2. 作业调度的时机是什么？进程调度的时机是什么？

3. 假设有一个系统中有5个进程，它们的到达时间和服务时间见表3-10，忽略I/O以及其他开销时间，分别按先来先服务、短进程优先、时间片轮转(注：时间片＝1)调度算法进行CPU调度，请给出各进程的完成时间、周转时间、带权周转时间，计算平均周转时间和平均带权周转时间。

表 3-10　进程的到达时间和服务时间

进　　程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

4. 假设有 4 个作业，它们的到达、运行时间见表 3-11，若采用高响应比优先调度算法，试问平均周转时间和平均带权周转时间为多少？（单位：小时，十进制）

表 3-11　4 个作业的到达、运行时间

作 业 号	到达时间	运行时间
1	8.0	2.0
2	8.3	0.5
3	8.5	0.1
4	9.0	0.4

5. 何谓死锁？产生死锁的原因和必要条件是什么？解决死锁的方法有哪些？

6. 设系统中有 3 类资源 A、B 和 C，又设系统中有 5 个进程 P1、P2、P3、P4 和 P5，在 T0 时刻系统状态见表 3-12，试问：

(1)给出 T0 时刻各进程的资源需求量 Need。

(2)系统是否处于安全状态？如是，则给出进程安全序列。

(3)如果进程 P5 申请 1 个资源类 A、1 个资源类 B 和 1 个资源类 C，能否实施分配？为什么？

表 3-12　进程和资源分配情况

进　　程	Max			Allocation			Available		
	A	B	C	A	B	C	A	B	C
P1	8	6	4	1	2	1	2	1	1
P2	4	3	3	3	1	1			
P3	10	1	3	4	1	3			
P4	3	3	3	3	2	2			
P5	5	4	6	1	1	3			

7. 试简化图 3-9 中的资源分配图，并利用死锁定理给出相应的结论。

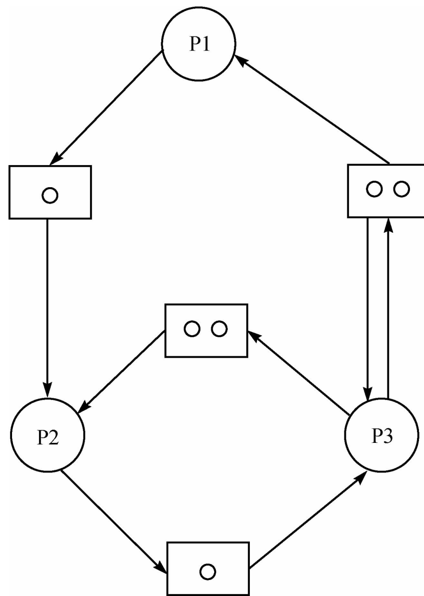

图 3-9　第 7 题图