进 程 管 理

7.1 进程的基本概念

7.1.1 程序的顺序执行和并发执行

1．程序的顺序执行

所谓程序的顺序执行是指该程序独占整个系统中的所有资源，处理机严格按照程序所规定的顺序进行操作，只有在前一个操作执行完后，才进行后继操作。

程序的顺序执行有以下特征。

（1）顺序性。

（2）封闭性。

（3）可再现性。

2．多道程序设计的引入

执行环境具有下述3个特点。

（1）独立性。

（2）随机性。

（3）资源共享。

3．程序的并发执行

程序的并发执行可总结为：一组在逻辑上互相独立的程序或程序段在执行过程中其执行时间在客观上互相重叠，即一个程序段的执行尚未结束，另一个程序段的执行已经开始的执行方式。

程序并发执行时具有如下特征。

（1）间断性。

（2）失去封闭性。

（3）不可再现性。

7.1.2 进程的定义和特征

1．进程的定义

进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动。

2．进程的特征

（1）结构特征

（2）动态性

（3）并发性

（4）独立性

（5）异步性

7.1.3 进程的状态及其转换

1．进程的基本状态

（1）就绪状态

当进程已分配到除处理机以外的所有必要的资源后，只要再获得处理机便可立即执行，这时进程的状态称为就绪状态。

(2）执行状态

执行状态是指进程已获得处理机、其程序正在执行的状态。

(3）阻塞状态

正在执行的进程因发生某事件而暂时无法继续执行时，便放弃处理机而处于暂停状态，这种暂停状态被称为阻塞状态。

2．进程的状态转换

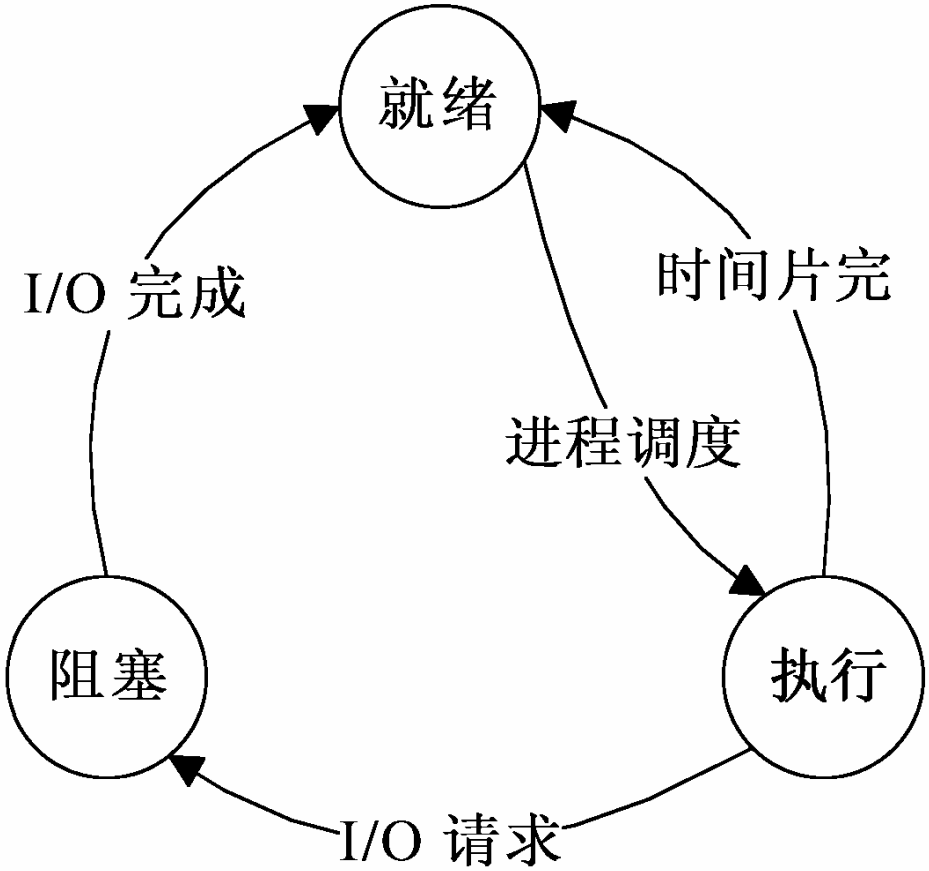

图7.1 进程的3种基本状态及其转换

7.1.4 进程的结构

1．进程的实体

（1）进程控制块（PCB）

（2）程序段

（3）数据段

2．进程控制块

进程控制块是进程实体的一部分，是操作系统中最重要的记录型数据结构。PCB中记录了操作系统所需的，用于描述进程进展情况及控制进程运行所需的全部信息。

PCB是进程存在的惟一标志。

一般把PCB存放在操作系统专门开辟的PCB区内。

在进程控制块中，主要包括下述4方面的信息。

(1）进程描述信息

- 进程标识符。每个进程都有惟一的进程标识符，用以识别不同的进程。
- 用户名或用户标识号。每个进程都隶属于某个用户，有利于资源共享与保护。
- 家族关系。标识进程之间的家族关系。

(2）处理机状态信息

通用寄存器、指令计数器、程序状态字（PSW）、用户栈指针等

(3）进程调度信息

- 进程状态。指明进程的当前状态，以作为进程调度和进程对换时的依据。

进程优先级。用于描述进程使用处理机的优先级别的一个整数，优先级别高的进程先获得处理机。

- 进程调度所需的其他信息。如进程已等待CPU的时间总和、进程已执行的时间总 和等。
- 事件。指进程被阻塞的原因。

(4）进程控制信息

- 程 序和数据的地址。指出该进程的程序和数据所在的内存或外存地址，以便再调度到该进程执行时，能从中找到其程序和数据。
- 进程同步和通信机制。指实现进程同步和进程通信时所必须的机制，如消息队列指针、信号量等。这些数据应全部或部分地存放在PCB中。

- 资源清单。它是一张列出了除CPU之外的进程所需的全部资源和已经分配给该进程的资源清单。
- 链接指针。它给出了本进程（PCB）所在队列的下一个进程的PCB首地址。

在一个系统中，通常拥有数十个、数百个乃至数千个PCB。为了对PCB进行有效地管理，系统应把所有的PCB用适当的方式组织起来。目前常用的PCB组织方式有链接方式和索引方式两种。

① 链接方式

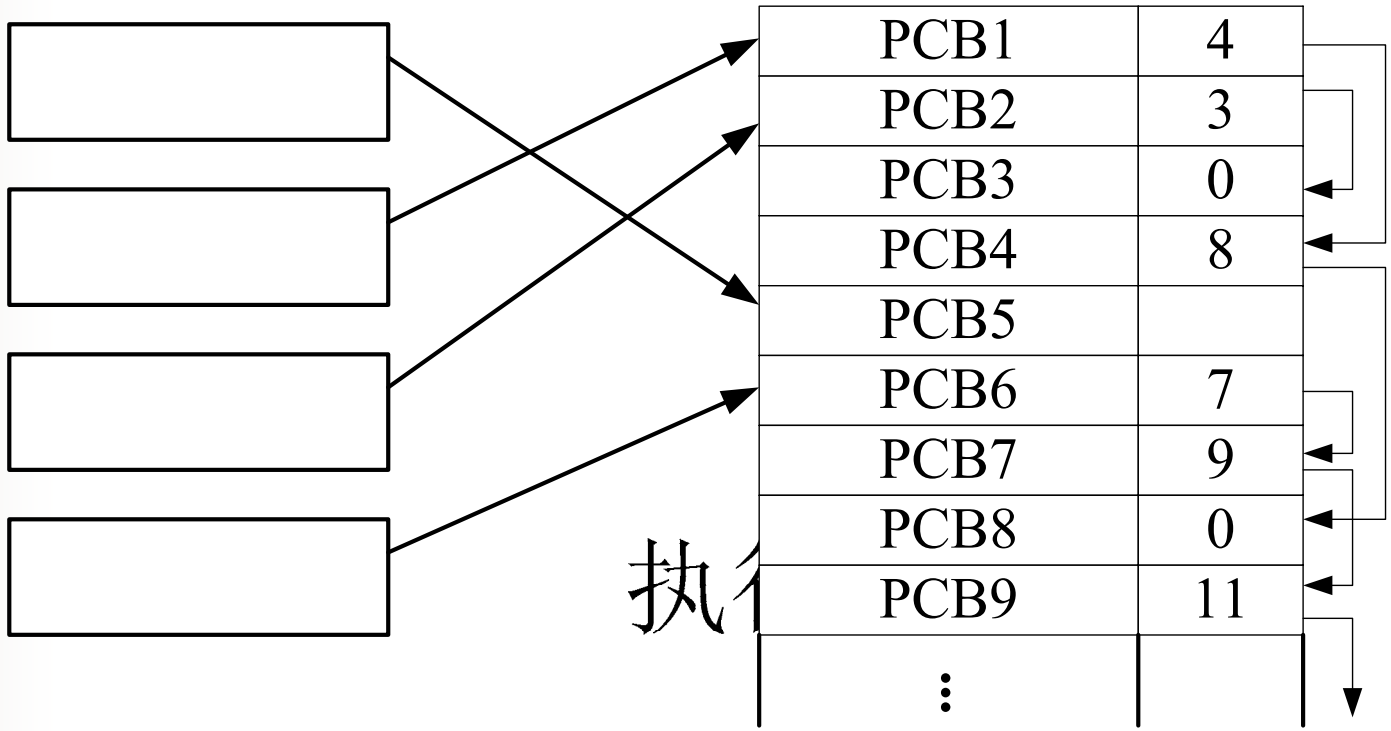

图7.2 PCB链接队列示意图

② 索引方式

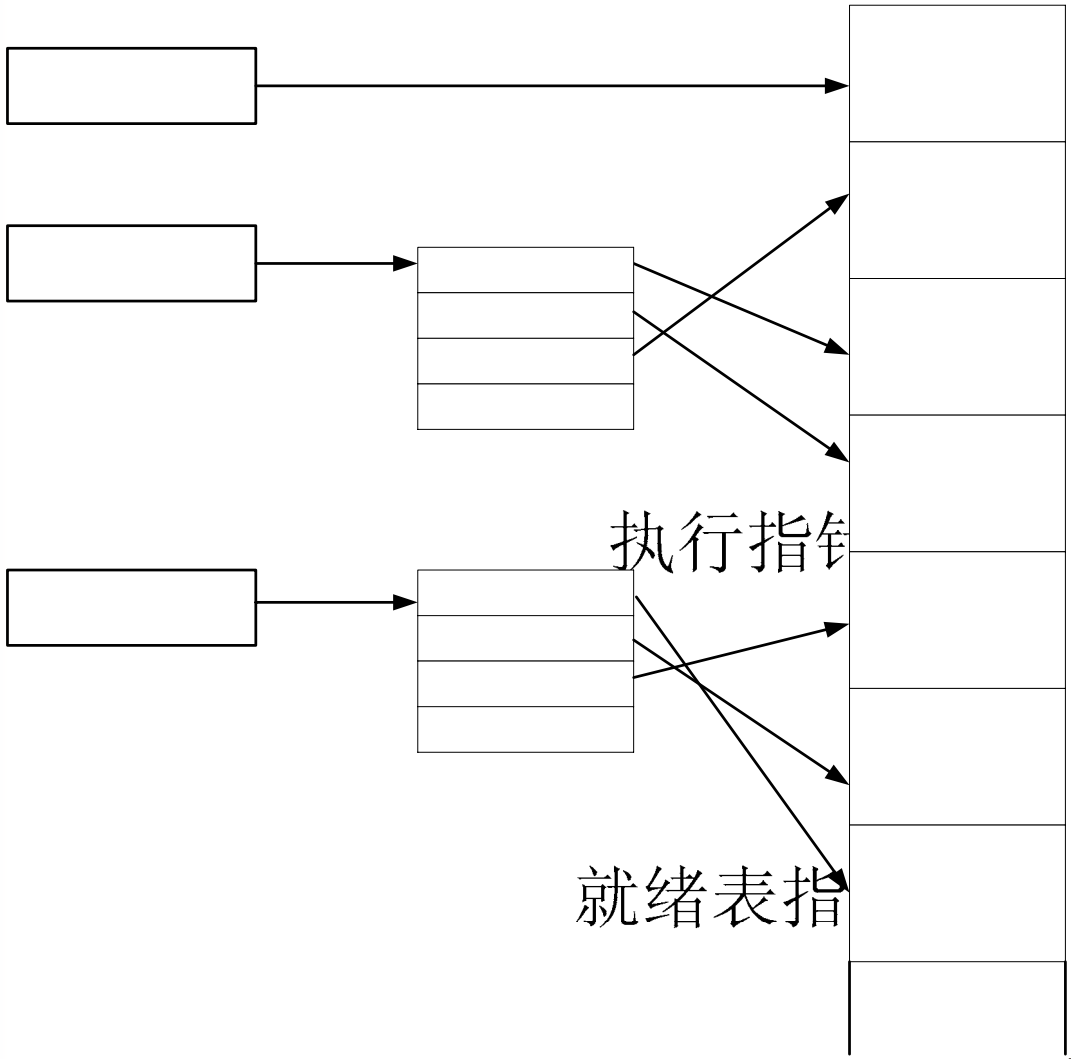

图7.3 PCB索引方式示意图

就绪索引

7.2 进程控制

7.2.1 操作系统内核

为了防止操作系统及关键数据（如PCB等）受到用户程序有意无意的破坏，通常将处理机的执行状态分成系统态和用户态两种。

（1）系统态。它具有较高特权，能执行一切指令，访问所有寄存器和存储区。

（2）用户态。这是具有较低特权的执行状态，只能执行规定的指令，访问指定的寄存器和存储区。

（3）操作系统内核：

在进行层次设计时，通常将一些与硬件紧密相关的模块，诸如中断处理程序、常用的设备驱动程序以及运行频率较高的模块（如时钟管理、进程调度和公共基本操作模块）都安排在紧靠硬件的软件层次中，并使之常驻内存，以提高操作系统的运行效率，通常把这一部分称为操作系统的内核（Kernel）。

7.2.2 进程控制的概念

进程控制是进程和处理机管理的一个重要任务。所谓进程控制，就是系统使用一些具有特定功能的程序段来创建、撤消进程以及完成进程在各种状态之间的转换，从而达到多进程高效率并发执行和协调资源共享的目的。

7.2.3 进程的创建与撤消

1．进程的创建

2．进程的撤消

7.2.4 进程的阻塞与唤醒

1．进程的阻塞

2．进程的唤醒

7.3 进程互斥和同步

7.3.1 进程互斥

1．互斥的概念

所谓进程互斥是指当有若干进程都要使用某一共享资源时，任何时刻最多允许一个进程使用，其他要使用该资源的进程必须等待，直到占用该资源者释放了该资源为止。

2．临界资源

操作系统中将一次只允许一个进程访问的资源称为临界资源。

3．临界区（Critical Section）

把进程中访问临界资源的那段程序代码段称为临界区。为实现对临界资源的互斥访问，应保证诸进程互斥地进入各自的临界区。必须在临界区前面增加一段用于进行上述检查的代码，我们把这段代码段称为进入区（Entry Section）；相应地，在临界区后面也要加上一段称为退出区（Exit Section）的代码，用于将临界区正被访问的标志恢复为未被访问的标志。

一个含有访问某一临界资源的循环进程可描述如下：

```
while(TRUE)
{
  entry section
  critical section
  exit section
  remainder section
}
```

4．同步机制应遵循的准则

（1）空闲让进。

（2）忙则等待。

（3）让权等待。

（4）有限等待。

7.3.2 进程同步

1．同步的概念

把异步环境下的一组并发进程因直接制约，使得各进程按一定的速度执行的过程称为进程间的同步。具有同步关系的一组并发进程称为合作进程，合作进程间互相发送的信号称为消息或事件。

2．同步与互斥的关系

进程的同步与进程的互斥都涉及到并发进程共享资源的问题，进程的互斥实际上是进程同步的一种特殊情况。

有时也把进程的互斥与进程的同步统称为进程的同步。

7.3.3 信号量机制

1．记录型信号量

```
struct semaphore
  {
      int value；
      PCB *L；
  }S；
```

2．P、V原语操作

除了给信号量S初始化外，信号量的数值域仅能由P、V原语操作改变（P和V分别是荷兰语Passeren和Verhoog的第一个字母，相当于英文的Pass和Increment的意思）。

P原语操作的主要动作是：

（1）S.value减1；

（2）若S.value减1后仍大于或等于零，则进程继续执行；

（3）若S.value减1后小于零，则该进程被阻塞，进入与该信号量相对应的等待队列L中，然后转进程调度。

P原语操作的功能框图如图7.8。

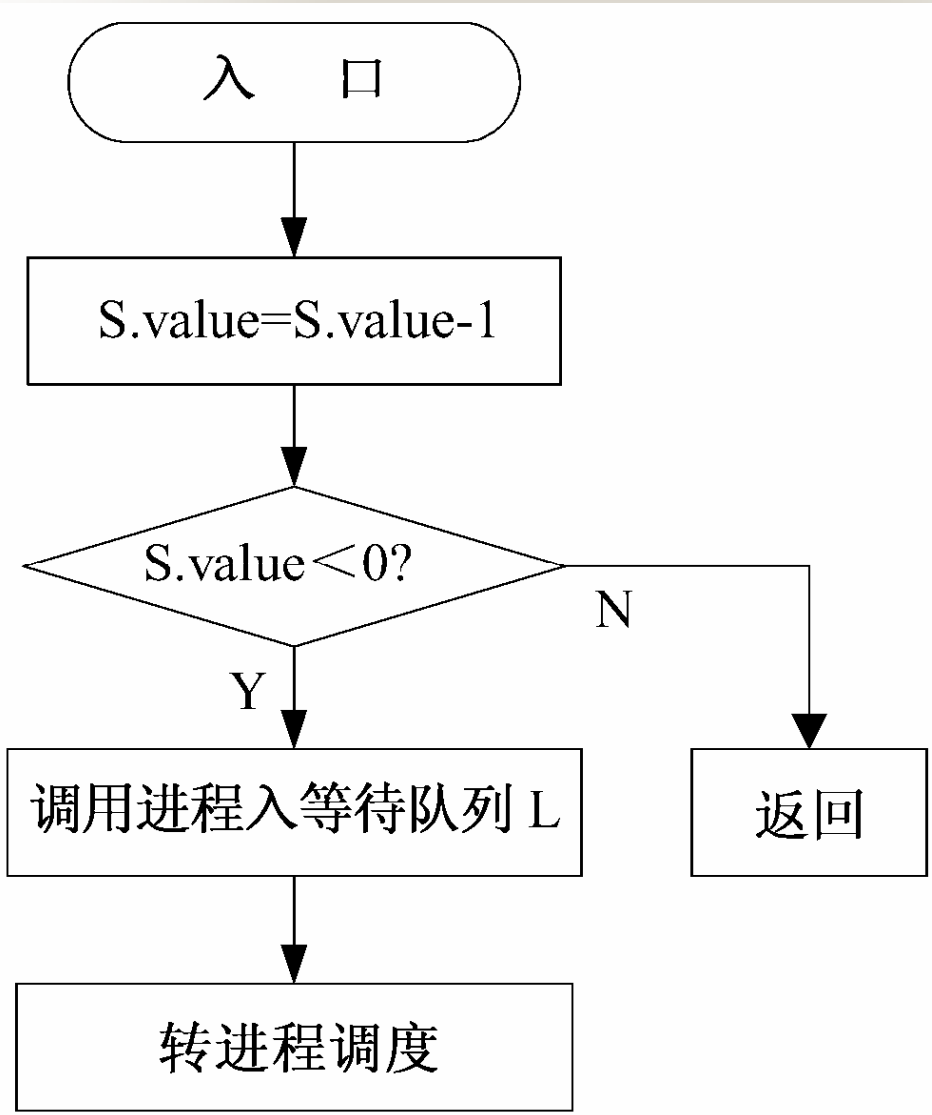

图7.8 P原语操作功能

V原语操作的主要动作是：

（1）S.value加1；

（2）若S.value加1后结果大于零，进程继续执行；

（3）若S.value加1后结果小于或等于零，则从该信号量的等待队列L中唤醒一个等待进程，然后再返回原进程继续执行或转进程调度。

V原语操作的功能框图如图7.9。

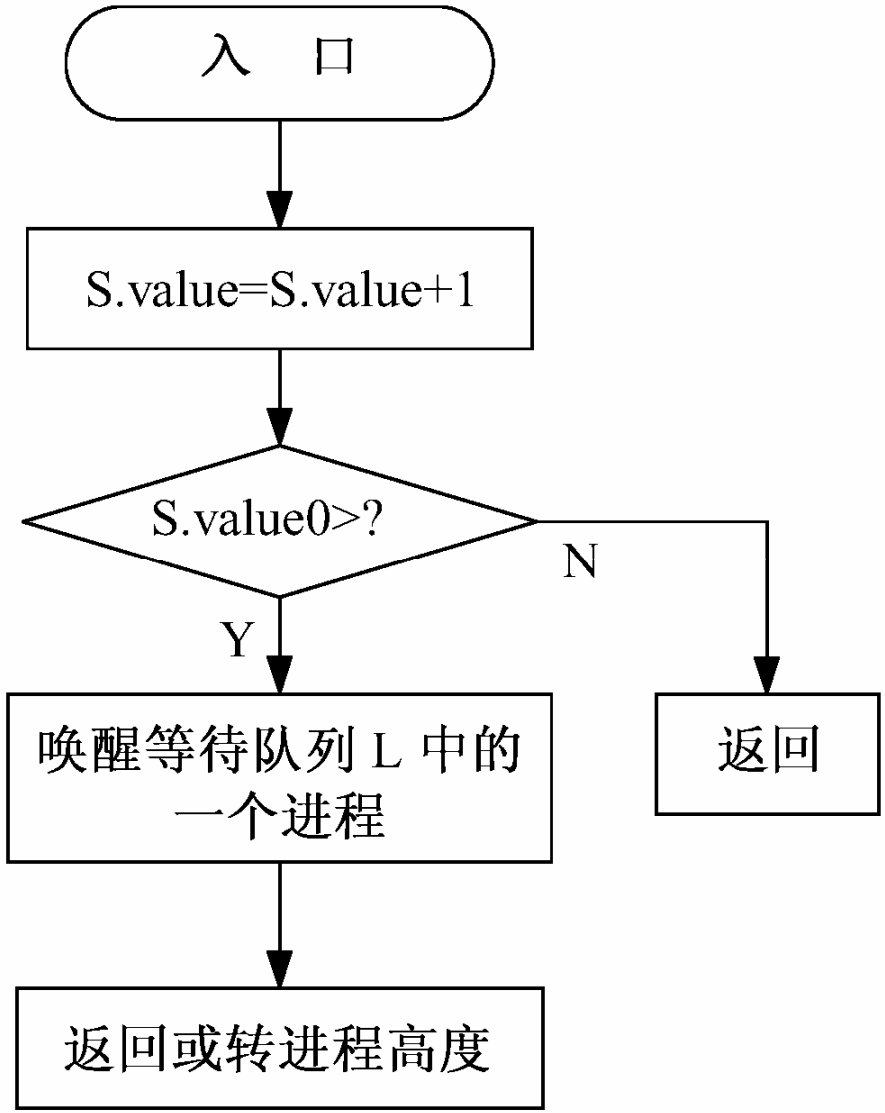

图7.9 V原语操作功能

7.3.4 进程互斥和同步的实现

1．进程互斥的实现

2．进程同步的实现

7.4 进程通信

通信（Communication）意味着在进程间传送数据。也把进程间控制信息的交换称为低级通信，而把进程间大批量数据的交换称为高级通信。

7.4.1 进程通信的类型

1．共享存储器系统

共享存储器系统为了传送大量数据，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进行读数据或写数据以实现通信。

2．消息传递系统

分为以下两种。

（1）直接通信方式

发送进程可将消息直接发送给接收进程，即将消息挂在接收进程的消息缓冲队列上，而接收进程可从自己的消息缓冲队列中取得消息。

（2）间接通信方式

发送进程将消息发送到指定的信箱中，而接收进程从信箱中取得消息。

3．管道通信系统

在UNIX操作系统中，开创了一种借助文件和文件系统形成的一种通信方式。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共享文件，又称pipe文件。向管道提供输入的发送进程，以字符流方式将大量的数据送入管道，而接收进程从管道中接收数据。由于发送进程和接收进程是利用管道进行通信的，故称为管道通信。

7.4.2 消息缓冲队列通信机制

1．消息缓冲队列通信机制简介

由于消息缓冲机制中所使用的缓冲区为公用缓冲区，因此使用消息缓冲机制传送数据时，两通信进程必须满足如下条件。

第一，在发送进程把写入消息的缓冲区挂入消息队列时，应禁止其他进程对该消息队列的访问，否则，将引起消息队列的混乱。同理，当接收进程正从消息队列中取消息时，也应禁止其他进程对该队列的访问。

第二，当缓冲区中无消息存在时，接收进程不能接收到任何消息；而发送进程是否可以发送消息，则只由发送进程是否能够申请到缓冲区决定。

2．消息缓冲队列通信机制中的数据结构

(1）消息缓冲区

```
typedefstruct message buffer
{
   sender；        //发送者进程标识符
   size；          //消息长度
   text；          //消息正文
   next；  //指向下一个消息缓冲区的指针
}
```

（2）PCB中有关进程通信的数据项

```
typedefstruct message block
{
  …
  mq；     //消息队列队首指针
  mutex；//消息队列互斥信号量，初值为1
  sm；     //消息队列资源信号量，用于消息队列中的消息计数，初值为0
  …
}
```

7.5 进 程 调 度

7.5.1 进程调度的概念

1．高级、中级和低级调度

（1）高级调度

高级调度通常也称作业调度，用于决定把外存上处于后备队列中的哪些作业调入内存，准备执行。

(2) 中级调度

中级调度大多针对于分时系统，是按一定的算法在内存和外存之间进行进程对换，目的在于缓和内存的紧张。

(3) 低级调度

低级调度用于将内存中就绪队列中的作业分配处理机，使其执行。

2．进程调度的方式

进程调度通常有以下两种方式。

（1）非剥夺方式

（2）剥夺方式

3．进程调度的功能

4．进程调度算法的性能评价

7.5.2 进程调度算法

1．先来先服务调度算法

在进程调度中，采用FCFS算法时，进程调度程序从就绪进程队列中，选择一个最先进入队列的进程，把处理机分配给它，让它进入执行状态。

公平性，并且实现也比较容易，这是它的优点。但是，它的缺点是实际上不公平，它比较有利于长进程，而不利于短进程。

2．短进程优先调度算法

短进程优先（SPF）调度算法，是指对执行时间短的进程优先调度的算法。SPF是从就绪队列中选出一个估计运行时间最短的进程，将处理机分配给它，使它立即执行并一直执行到完成，或因等待某事件发生而放弃处理机时，再重新调度。

采用SPF算法，平均周转时间比FCFS调度算法有很多改善，这是它的优点。SPF调度算法的缺点是如下所述。

第一，对长进程非常不利。

第二，紧迫进程不能及时处理。

第三，执行时间的估计值不准确。

3．高优先级优先调度算法

考虑到系统中的紧迫进程能得到优先处理，引入了高优先级优先（HPF）调度算法，处理机总是分配给就绪进程队列中优先级最高的进程。

进程的优先级可采用静态优先级和动态优先级两种，优先级可由用户自定或由系统确定。

4．时间片轮转法

时间片轮转法的基本思想是将CPU的处理时间分成固定大小的时间片。如果一个进程在被调度选中之后用完了系统规定的时间片，但未完成要求的任务，则它自行释放自己所占有的CPU，而排到就绪队列的末尾，等待下一次调度。同时，进程调度程序又去调度当前就绪队列中的第一个进程。

5．多级反馈队列调度算法

其基本思想如下所述。

（1）系统按优先级设置N个就绪进程队列，第一级队列的优先级最高，其余队列的优先级逐个降低，第N级队列的优先级最低。

7.6 死　锁

在多道程序系统中，多个进程并发执行，共享系统资源，从而提高了资源利用率和系统吞吐量，但可能发生一种危险——死锁。所谓死锁，是指多个进程因竞争资源而形成一种僵局，若无外力作用，这些进程都将永远不能再向前推进。

7.6.1 产生死锁的原因和必要条件

1．产生死锁的原因

产生死锁的主要原因可归结为以下两点。

（1）竞争资源

（2）进程推进顺序不当

2．产生死锁的必要条件

(1）互斥条件

一个资源在一段时间内只能被一个进程所使用，具有排它性。

(2）请求和保持条件

一个进程在请求新资源而阻塞时，对已获得资源又保持不放。

(3）不剥夺条件

进程已获得的资源，在未使用完之前不能被剥夺，只能在使用完时由自己释放。

(4）环路等待条件

在发生死锁时，必然存在一个进程——资源的环形链。即进程集合{P1，P2，…，P*n*}中的P1正在等待P2占用的资源，P2正在等待P3占用的资源，…，P*n*正在等待P1占用的资源。

只要同时具备上述4个必要条件，系统就会发生死锁，只要上述条件之一不满足，系统就不会发生死锁。

3．处理死锁的方法

由于死锁状态的出现会给整个系统带来严重的后果，所以如何解决死锁问题引起了人们的普遍关注。目前常用的方法有以下3种。

（1）预防死锁

为了使系统中不发生死锁现象，在系统设计初期即选择一些限制条件，来破坏产生死锁的4个必要条件之一或其中几个。这样，系统中就不会出现死锁现象。这种方法对预防死锁的发生非常有效，但有可能降低系统资源的利用率。

（2）避免死锁

由于一方面预防死锁的方法会降低系统资源利用率，另一方面死锁的必要条件的存在未必就一定会使系统发生死锁，因此为提高系统资源的利用率，可采用避免死锁。避免死锁并不严格限制死锁必要条件的存在，而是在资源的动态分配过程中，使用某种方法去防止系统进入不安全状态，从而避免死锁的最终出现。

（3）检测和解除死锁

由于死锁产生的概率总是比较小的，所以在一些相对简单的系统中，为节省预防或避免死锁中所增加的系统开销，系统中允许出现死锁状态。在这种系统中，专门设置了一个检测机构，可以随时检测出死锁的发生，并能确定与死锁有关的进程和资源，然后采用适当的方法解除系统中的死锁状态。

常用的解除死锁的方法有两种：一是强制性地撤销一些死锁进程，并剥夺它们的资源给其他的进程；另一种是使用一个有效的挂起和解除挂起机构来挂起一些进程，以便从被挂起进程中剥夺一些资源，用来解除死锁。

7.6.2 预防死锁

1．打破“请求和保持”条件

打破“请求和保持”条件，即把进程运行中所要求的所有资源在进程开始运行之前，一次性地分配给进程，只要有一种资源不能满足，该进程就必须等待。这样，进程在运行过程中就不再需要新的资源，这种方法又称为预先静态分配法。

2．打破“不剥夺”条件

打破“不剥夺”条件，即强迫那些请求新资源而没有立即得到满足的进程释放它已保持的其他资源。这意味着一个进程在运行过程可以暂时释放已占有的资源，即允许其他进程剥夺使用该资源，从而破坏了“不剥夺”条件的出现。

3．打破“环路等待”条件

死锁产生时，一定存在一种进程和资源的循环链。打破“环路等待”条件就是在资源的分配过程中，对资源的请求做出某种限制，使环路不可能出现。

7.7 线　程

7.7.1 线程的引入

由于进程是一个资源拥有者，所以在进程的创建、撤消和调度切换以及进程的同步与通信中，系统必须付出较大的时空开销。正因为如此，在系统中所设置的进程数目不宜过多，进程切换的频率也不宜过高，这也就限制了并发程度的进一步提高。

由以上对进程的分析可知，如果将进程的上述两个属性分开，由操作系统分开处理，将使多个程序更好地并发执行，同时又可减少系统的开销。也就是说，对于作为调度和分派的基本单位，不同时作为拥有资源的单位；而对于拥有资源的基本单位，又不对之进行频繁的切换。正是在这种思想的指导下，形成了线程（Thread）的概念。

在引入线程的操作系统中，线程是进程中的一个实体，是被系统独立调度和分派的基本单位。它的执行环境很小，除了自身必需的堆栈、寄存器、优先级等私有资源外，共享其所属进程的资源。

7.7.2 线程与进程的比较

(1）拥有资源

(2）调度

(3）并发性

(4）系统开销

7.7.3 线程的属性

线程具有如下属性。

（1）线程有控制表。

（2）线程共享所属进程的资源。

（3）线程是处理机的独立调度单位，多个线程可以并发执行。

（4）线程有动态性。

7.7.4 线程的状态及其转换

（1）就绪状态。线程已具备了执行的条件，等待线程调度程序调度。

（2）备用状态。由调度程序选定为一个执行对象。

（3）转换状态。若线程已准备好执行，但突然资源不可用，从而成为转换状态。

（4）运行状态。获得CPU正在执行。

（5）等待状态。正在执行的线程，由于某种原因（如I/O操作）不能继续运行下去。

（6）终止状态。线程已执行完成。

线程的状态及其转换如图7.14所示。

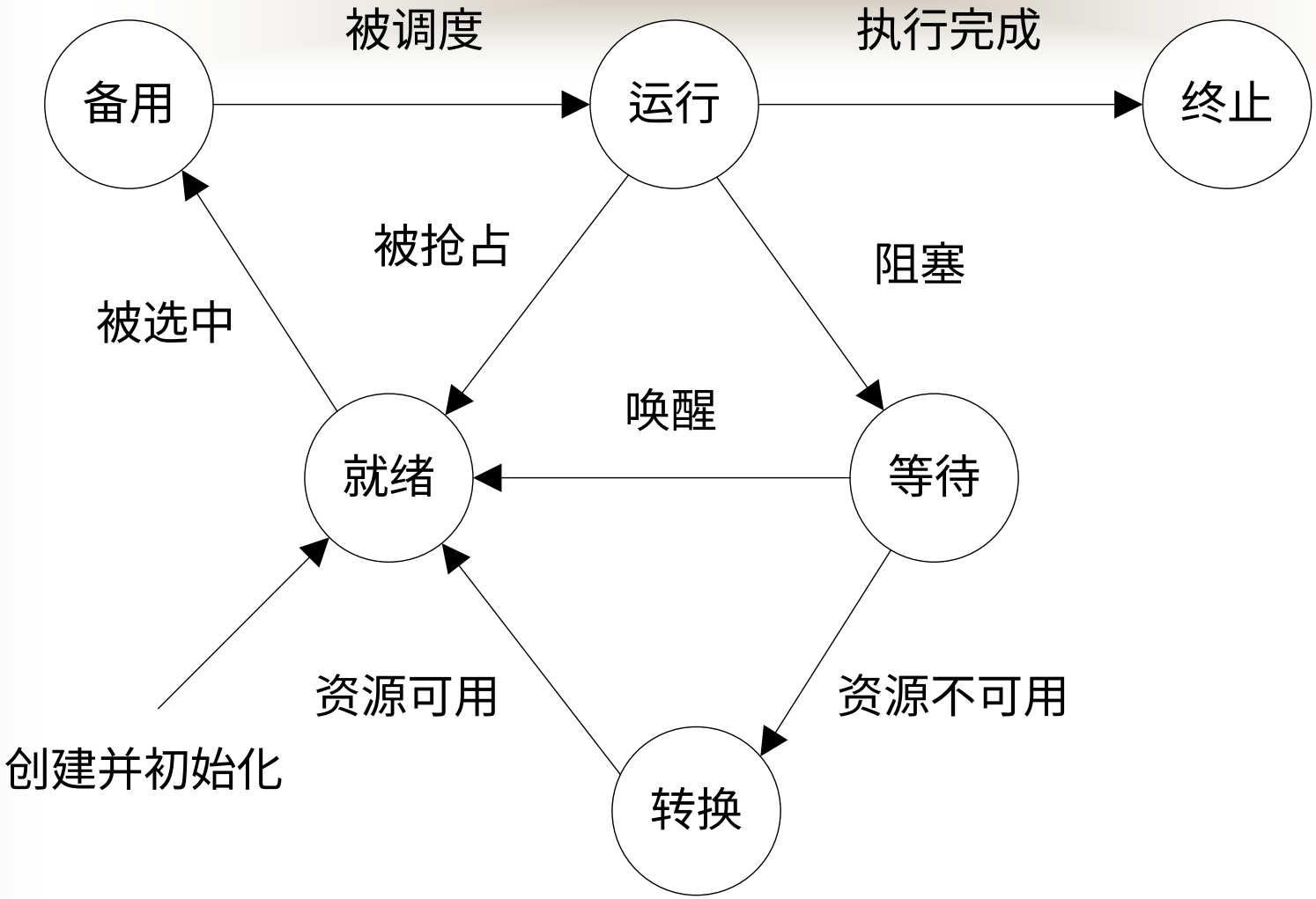

图7.14 线程的状态及其转换

7.8 Linux中的进程管理

7.8.1 Linux进程概述

1．进程实体的组成

Linux进程由3部分组成：正文段、用户数据段和系统数据段，如图7.15所示。

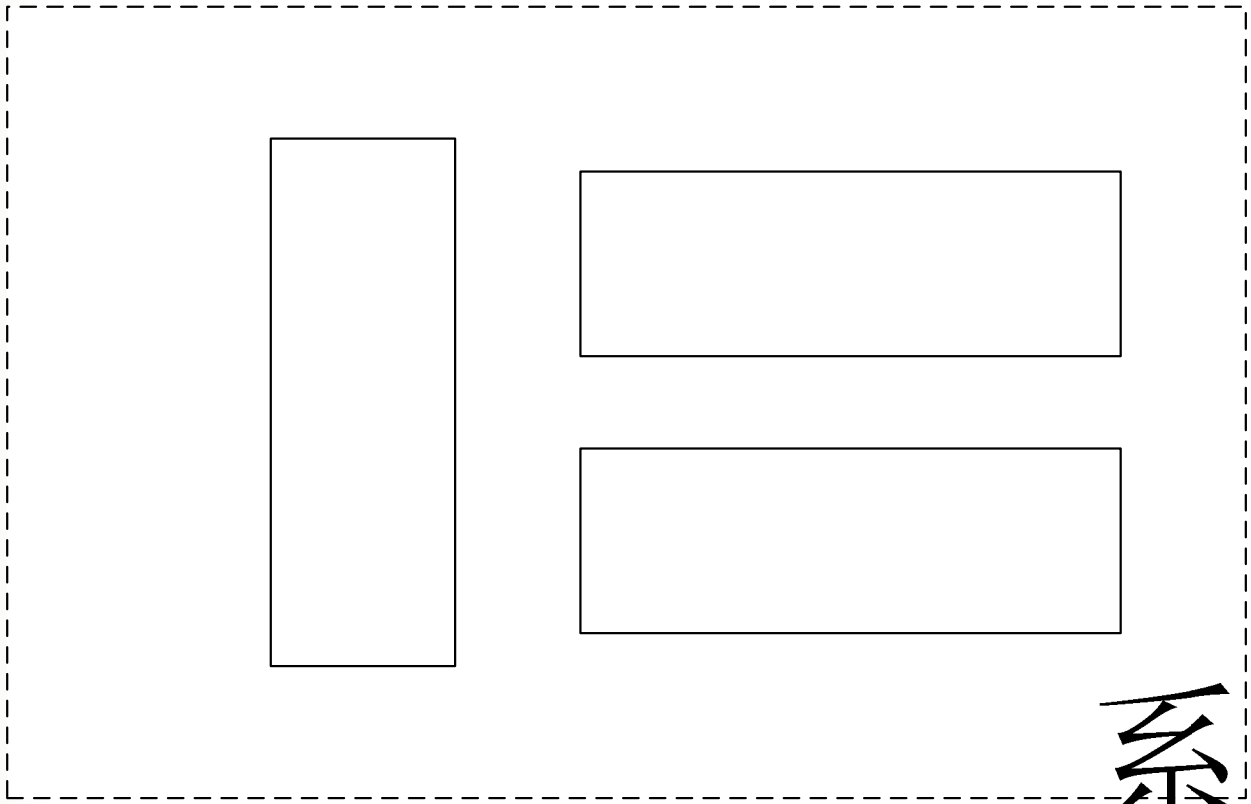

图7.15 Linux进程组成

2．进程的状态

进程是一个动态的概念，在其运行的整个生命周期中可根据具体情况不断改变其状态。Linux进程主要有如下几种状态。

（1）运行状态（task_running）

（2）等待状态

（3）暂停状态（task_stopped）

（4）僵死状态（task_zombie）

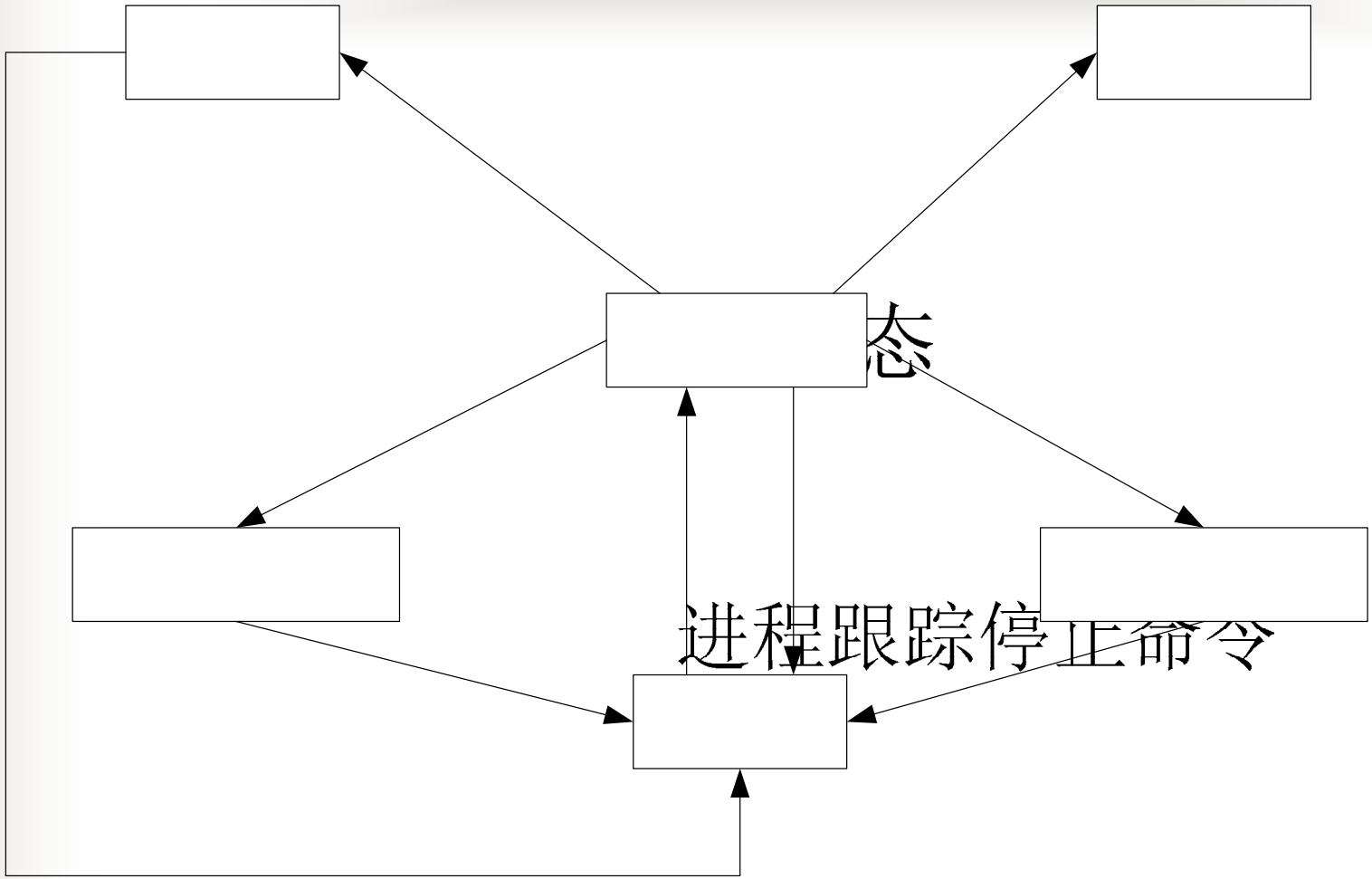

图7.16 Linux进程状态转换

被 未申请到所需资源

唤

7.8.3 Linux的进程调度

7.8.4 Linux进程的同步和通信

1. 信号机制
2. 管道机制
3. 消息队列
4. 共享内存
5. 信号量