第二章 进程管理

一、教学目的要求：

1．掌握进程的引入

2．掌握进程的定义以及特征

3．掌握进程与程序的区别

4.了解进程的静态描述包括的元素

5．掌握进程控制块

6．掌握进程的状态以及转换

7．掌握原语的概念

8．掌握几种进程控制原语

9．了解 UNIX 操作系统中的进程状态及一些简单的进程管理命令

10．掌握临界区的概念

11．掌握同步和互斥的概念

12．掌握同步和互斥的实现

13．会根据实际问题使用 P,V 操作

14．掌握管程的概念并了解其实现过程

15．掌握消息缓冲的概念并了解其实现过程

16．掌握线程的概念

二、内容分析：

1．概述：

1）进程是操作系统中的一个核心概念，对学生来讲也是一个新的概念，其引入一定要讲清楚。把引入讲清楚了，进程的其它方面就可以容易地展开。

2）系统是通过进程控制块识别进程的，进程控制块是一个重要概念。进程的动态特征的体现是进程的状态及其转换。

3）结合进程状态分析进程控制原语，注意原语与一般程序的区别。

4）同步和互斥是进程通信中两个非常基本的且很重要的概念。引入时要通过实例来说明。另外对它们的控制也很重要，可以通过一些算法来说明。

5）管程和消息缓冲通信作为扩充内容来讲，其主要目的还是巩固同步和互斥。

6）引入进程是为了实现并发，引入线程是为了提高并发程度，减少系统开销。

2．教学重点：

1）进程的引入

2）进程的定义

3）进程控制块

4）进程状态及其转换

5）原语的概念

6）四种进程控制原语

7）临界区的概念

8）同步和互斥的概念及实现

9）消息缓冲的概念

10）线程的概念及状态

3．教学难点：

1）进程的引入

2）进程状态及转换

3）原语实现

4）同步与互斥的实现

5）线程的概念

进程的引入

1． 程序的顺序执行

顺序程序活动有三个主要特点：

(1) 程序所规定的动作在机器上严格地按顺序执行。

(2) 只有程序本身的动作才能改变程序的运行环境。

(3) 程序的执行结果与程序运行的速度无关。

上述特点概括起来就是程序封闭性和可再现性。所谓封闭性就是指程序一旦运行起来，其计算结果仅取决于程序本身;除了人为地改变运行状态或机器出现故障外,没有别的因素能影响程序运行的过程。所谓可再现性就是当机器在同一数据集上重复执行同一程序，每次执行都会得到相同的结果。

程序顺序执行的特征：顺序性、封闭性、可再现性。

图 2.1 顺序处理的操作次序

2. 多道程序设计

多道程序设计是在一台计算机上同时运行两个或更多个程序。从宏观上看，系统中的多个程序都同时得到执行，即程序是并发执行的。多道程序设计具有提高系统资源利用率和增加作业吞吐量的优点。

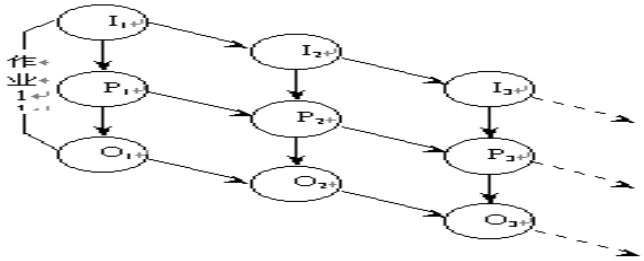


图 2.2　并行计算的先后顺序

例：（一个极端化的例子，但能说明问题）

假定有两道作业 A 和 B 都在执行，每个作业都是执行一秒钟，然后等待一秒钟，进行数据输入，随后再执行，再等待……一直重复 60 次。如果按单道方式，先执行作业 A，A 作完了再执行 B，那么两个作业都运行完，共需要 4 分钟，CPU 的利用率为百分之五十。如果我们采用多道程序技术来执行同样的作业 A 和 B 就能大大改进系统性能，作业 A 先运行，它运行一秒后等待输入。此时让 B 运行，B 运行一秒后等待输入，此时恰好 A 输入完，可以运行了，……就这样在 CPU 上交替地运行 A 和 B，在这种理想的情况下，CPU 不空转，其使用率升到百分之百，并且吞吐量也随之增加了。

小结：多道程序设计——多个程序同时在内存并且运行。

多道程序设计的特点：多个程序共享系统资源，多个程序并发执行。

多道程序设计的优点：提高资源利用率，增加系统吞吐量。

3.　程序并发执行时的特征

1）失去封闭性

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”的概念。

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相互制约。

4.进程概念的引入

由于多道程序并发执行时共享系统资源，共同决定这些资源的状态，因此系统中各程序在执行过程中就出现了相互制约的关系，程序的执行出现“走走停停”的新状态。这些都是在程序的动态过程中发生的。而程序本身是机器能够翻译或执行的一组动作或指令，是静止的。因此，用程序这个静态概念已不能如实反映程序并发执行过程中的这些特征。为此，人们引入“进程”（Process）这一概念来描述程序动态执行过程的性质。

进程的定义

进程最基本的属性是动态性和并发性。

进程定义为：**程序在并发环境中的执行过程**。

进程和程序是两个完全不同的概念，但又有密切的联系。它们的主要区别是：

（1）程序是静态概念，本身可以作为一种软件资源长期保存着；而进程则是程序的一次执行过程。它是动态概念，有一事实上的生命期，是动态地产生和消亡的。

（2）进程是一个能独立运行的单位，能与其它进程并发执行。进程是作为资源申请和调度单位存在的；而通常的程序是不能作为一个独立运行的单位而并发执行的。

程序在 CPU 上才能得到真正的执行。系统中是以进程为单位进行 CPU 的分配。

（3）程序和进程无一一对应关系。一个程序可由多个进程共用；另一方面，一个进程在其活动中又可顺序地执行若干个程序。

（4）各个进程在并发执行过程中会产生相互制约关系。

进程的特征

进程具有以相特征：

（1）动态性：进程是程序的执行过程，它有生有亡，有活动有停顿，可以处于不同的状态。（2）并发性：多个进程的实体能存在于同一内存中，在一段时间内都能运行。

（2）调度性：进程是系统中申请资源的单位，也是被调度的单位。

（3）异步性：各进程向前推进的速度是不可预知的，即异步方式运行。

（4）结构性：进程有一定的结构，它由程序段、数据段和控制结构等组成。

进程与程序的主要区别：

（1）进程是动态的；程序是静态的；

（2）进程是独立性的，能并发执行的；程序不能并发执行；

（3）二者无一一对应关系；

（4）进程异步运行，会相互制约；程序不具备此特征；

进程的组成

1. 进程的组成

用来描述进程当前的状态、本身的特性的数据结构被称为进程控制块（Process Control Block,简称 PCB），所以进程实体通常由程序、数据集合和 PCB 这三部分组成。

进程的这三部分构成进程在系统中的存在和活动的实体，有时也统称为“进程映象”。

2. 进程控制块的组成

进程控制块有时也称进程描述块（Process Descriptor）它是进程组成中最关键的部分。其中含有进程描述信息和控制信息，是进程动态性的集中反映，它是系统对进程进行识别和控制的依据。为了刻画进程的动态变化，通常把进程表示为由程序段、私有数据块和进程控制块组成，如图 2.3（a）所示。程序部分描述进程本身所要完成的功能，而“私有数据块”是接受程序规定操作的一组存储单元的内容，是操作的对象。进程控制块是在进程创建时产生的，当进程存在于系统时（运行），进程控制块就标识了这个进程。如图 2.3（b）所示。

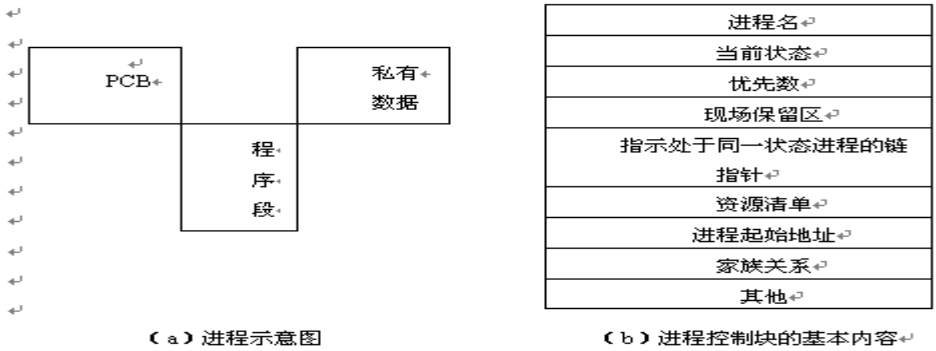


图 2.3 进程控制块

一般来说，进程控制块包括如下内容：

（1）进程名：它是惟一的标志对应进程的一个标志符或数字；

（2）特征信息：包括是系统进程还是用户进程；

（3）进程状态信息：表明该进程的执行状态；

（4）调度优先权：表示进程获取 CPU 的优先级别；

（5）通信信息：反映该进程与哪些进程有什么样的通信关系；

（6）现场保护区：被保护的信息有：程序计数器程序状态字，各工作寄存器的内容等；

（7）资源需求、分配和控制方面的信息；

（8）进程实体信息：指出该进程的程序和数据的存储情况，在内存或外存的地址、大小等；

（9）族系关系：反映父子进程的隶属关系；

（10）其它信息：如文件信息、工作单位等。

3．进程控制块的作用

进程控制块是进程存在的标志，当系统或父进程创建一个进程时，实际上就是为其建立一个进程控制块。 进程控制块既能标识进程的存在，又能刻画出进程的动态特征，它是一个进程仅有的被系统真正感知的部分。对操作系统而言，所有进程控制块将构成并发执行控制和维护系统工作的依据。

进程的状态和转换

进程的动态性是由它的状态和转换体现出来的。

1. 进程的基本状态

三种基本状态是：运行态、就绪态和阻塞态（或等待态）

（1）运行态（Running）

运行状态是指当前进程已分配到 CPU，它的程序正在处理机上执行时的状态。

（2）就绪态（Ready）

就绪状态是指进程已具备运行条件，但因为其它进程正占用 CPU，所以暂时不能运行而等待分配 CPU 的状态。

（3）阻塞态（Blocked）

阻塞态是指进程因等待某种事件发生而暂时不能运行的状态。

2．进程的状态及其转换：

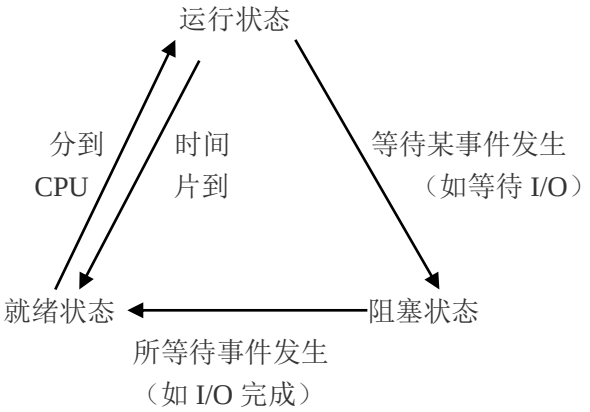


图 2.4 进程状态转换

（1）就绪——运行

处于就绪状态的进程被调度程序选中，分配到 CPU 后，该进程的状态就由就绪态变为运行态。

（2）运行——阻塞

正在运行的进程因某个条件未满足而放弃对 CPU 的占用，这个进程的状态就由运行态变为阻塞态。

（3）阻塞——就绪

处于阻塞状态的进程所等待事件发生了，系统就把该进程的状态由阻塞态变为就绪态。

（4）运行——就绪

正在运行的进程如用完了本次分配给它的 CPU 时间片，它就得从 CPU 上退下来,暂停运行。该进程状态就由运行态变为就绪态.

小结: 进程基本状态

运行态：此时正用 CPU；

就绪态：可运行,但未分到 CPU；

阻塞态：不能运行,等待某个外部事件发生。

在一定条件下，进程状态才发生转换。

进程队列的概念

为了对所有进程进行有效的管理，常将各进程的 PCB 用适当的方式组织起来，一般有以下几种方式：线性方式、链接方式和索引方式。详见教材 P47—48。

进程的管理和有关命令

进程是有“生命期”的动态过程，核心能对它们实施管理，这主要包括：创建进程、撤消进程、挂起进程、恢复进程、改变进程优先级、封锁进程、唤醒进程、调度进程等。

1．进程的管理

1）进程族系

就如同人类的族系一样，系统中众多的进程也存在族系关系：由父进程创建子进程，子进程再创建子进程，……，从而构成一棵树型的进程族系图，如下图

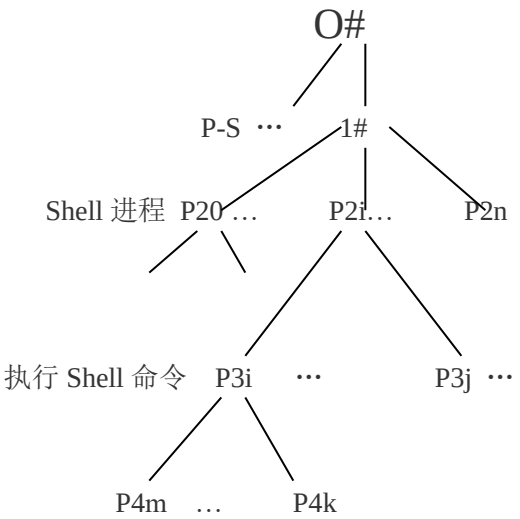


图 2.5 进程创建的层次关系

原语

所谓原语（Primitive）是机器指令的延伸，往往是为了完成某些特定的功能而编制的一段系统程序。为保证操作的正确性，在许多机器中规定，执行原语操作时，要屏蔽中断，以保证操作的不可分割性，即一个操作中的所有动作要么全做，要么全不做。操作系统中完成某些基本操作时，往往利用原语操作来实现。

内核中有很多原语，如创建进程、终止进程、阻塞进程等。

进程控制原语

1．进程创建

进程创建原语的主要操作过程是：

（1）申请一个空闲的 PCB；

（2）为新进程分配资源；

（3）将进程的 PCB 初始化；

（4）将新进程加到就绪队列中。

2．进程终止

终止进程的主要操作过程是：

（1）从系统的 PCB 表中找到指定进程 PCB；

（2）回收该进程所占用的全部资源；

（3）若该进程还有子孙进程，则还要终止其所有子孙进程，回收它们所占用的全部资源；

（4）释放被终止进程的 PCB，并从原来队列中摘走。

3．进程阻塞

正在运行的进程通过调用阻塞原语主动地把自己阻塞，阻塞的过程如下：

（1）立即停止当前进程的执行；

（2）将现行进程的 CPU 现场送到该进程的 PCB 现场保护区中保存起来，以便将来重新运行时恢复此时的现场。

（3）把该进程 PCB 中的现行状态由“运行”改为阻塞，把它插入到具有相同事件的阻塞队列中。

（4）然后转到进程调度程序，重新从就绪队列中挑选一个合适进程投入运行。

4．进程唤醒

唤醒原语执行过程：

（1）首先把被阻塞进程从相应的阻塞队列中摘下；

（2）将现行状态改为就绪态，然后把该进程插入到就绪队列中；

（3）如果被唤醒进程比运行进程有更高的优先级，则设置重新调度标志。

阻塞原语与唤醒原语恰好是一对相反的原语：调用前者（阻塞原语）是自己去睡眠，调用后者（唤醒原语）是把“别人”唤醒。使用时也要成对，前边有睡的，后边有叫醒的。否则，前者就要“长眠”了。如同两个士兵轮流站岗值班和睡觉休息。相关者唤醒，自己不能唤醒自己。

进程管理命令

1. UNIX 系统中进程状态

在 UNIX 系统中，进程状态可分为 10 种，它们是（进程的一般状态）：

（1）用户态运行（在 CPU 上执行用户程序）；

（2）核心态运行（在 CPU 上执行核心程序）；

（3）在内存就绪（具备运行条件，只等核心调度它取得 CPU 控制权）；

（4）在内存睡眠（不具备运行条件，在内存中等待某一事件发生）；

（5）在外存就绪（就绪进程被对换到外存上）；

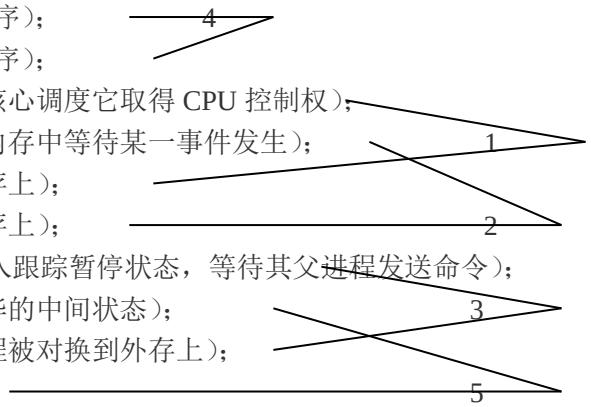
（6）在外存睡眠（睡眠进程被对换到外存上）；

（7）在内存暂停（因调用 stop 程序而进入跟踪暂停状态，等待其父进程发送命令）；

（8）创建态（新进程被创建，但尚未完毕的中间状态）；

（9）在外存暂停（处于跟踪暂停态的进程被对换到外存上）；

（10）终止态（进程终止自己）。



上述十种状态可分为五组（如上图示）。

2. UNIX 系统中进程映象

UNIX 进程映象由以下部分组成：PCB、进程执行的程序，进程执行时所用的数据、进程运行时使用的工作区。

3. PS 命令

PS 命令用来检查系统中当前存在的进程的状态。

4. Sleep 命令

Sleep 命令使进程暂停执行一段时间。

5. 后台命令符 &

若在命令行的末尾加上&字符，该命令将在后台执行。在没有前台进程运行时，它才得以执行。

6. Wait 命令

Wait 命令的功能是等待后台进程结束。

7. kill 命令

kill 命令的功能是终止一个进程的运行。

例如在超级用户方式下，

kill678 ， 则停止 PID 为 678 的进程运行。

8. nice 命令

其功能是以不同的优先级执行一条命令。

临界资源和临界区

在计算机中有许多资源只允许一个进程使用，如果有多个进程同时去使用这类资源就会产生严重的错误。

几个进程若共享同一临界资源，它们必须以互斥的方式使用这个临界资源，即当一个进程正在使用临界资源且尚未使用完毕时，则其他进程必须推迟对该资源的进一步操作，在当前进程的使用完成之前，不能从中插进去使用这个临界资源，否则将会造成信息混乱和操作出错。

系统中同时存在有许多进程，它们共享各种资源，然而有些资源每次只能让一个进程所使用。

同步与互斥

1．同步：我们把进程间的这种必须互相合作的协同工作关系，称为进程同步。

2．互斥：两个并行的进程 A、B，如果当 A 进行某个操作时，B 不能做这一操作，进程间的这种限制条件称为进程互斥，这是引起资源不可共享的原因。

同步与互斥的实现

lock 和 unlock

大部分同步方案均采用某个物理实体（如锁、信号灯等）实现通信，进程通信原语中关锁（lock）和开锁（unlock）是最简单的原语。在这两个原语中设置一个公共变量 x 代表某个临界资源的状态。如：x=0，表示资源可用，x=1，表示资源正在使用。

关锁原语 lock（x）：　　L：if　x＝1 then　goto　L

　　else　x：=1；

开锁原语 unlock（x）：x：=0；

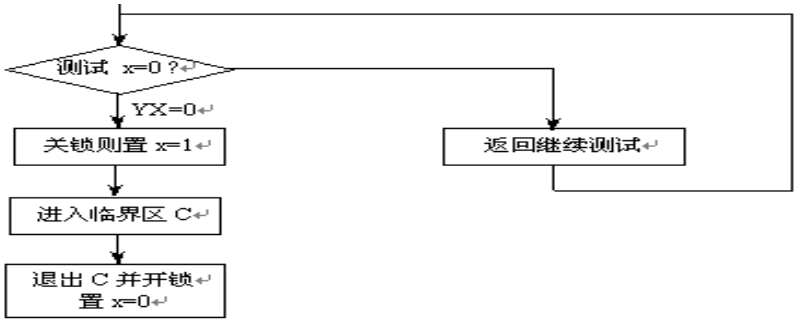


图 2-6 lock 和 unlock 流程图

两个经典的同步、互斥例子

1．生产者与消费者问题

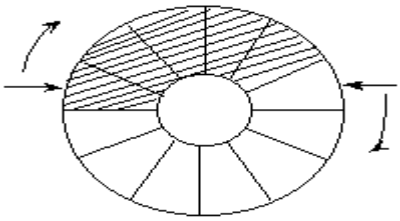
Dijkstra 把广义同步问题抽象成一种“生产者与消费者问题（Producer-consumer-relationship）的抽象模型。事实上，计算机系统中的许多问题都可归结为生产者与消费者问题，生产者与消费者可以通过一个环形缓冲池（见图 3.8）联系起来，环形缓冲池由几个大小相等的缓冲块组成，每个缓冲块容纳一个产品。每个生产者可不断地每次往缓冲池中送一个生产产品，而每个消费者则可不断地每次从缓冲池中取出一个产品。

下面给出基于环形缓冲区的生产者与消费者关系的形式描述，设：（1）公用信号量 mutex：初值为 1，用于实现临界区互斥。（2）生产者私用信号量 empty：初值为 n，指示空缓冲块数目。（3）消费者私用信号量 full：初值为 0，指示满缓冲块数目。（4）整型量 i 和 j 初值均为 0，i 指示首空缓冲块序号，j 指示首满缓冲块序号。

模块 设计如下：

```
Var mutex，empty，full：semaphore;
    i,j:integer;
```

```
    buffer：array［0…n 一 1］of  item;
Procedure  producer;  生产者进程
 begin
while  true  do
  begin
  produce a product;
```



```
    P(empty);
     P(mutex);
     Buffer (i)：＝Product;
     i：＝(i+1) mod  n；
     V(mutex);
     V(full)
          end
    end;
    procedure consumer;      消费者进程
begin
          While true do
          begin
            P(full)；
             P(mutex)
            goods：＝buffer(j);
             j：＝(j+1) mod  n;
           V(mutex);
           V(empty);
           Consume a product;
        end
  end;
begin
    seminitial ;
    i:＝j:＝0;
    cobegin
    producer；
     consumer；
 coend
 end
```

2．读者与写者问题

设某航空公司有 2 个售票处，它们通过远程终端访问设在公司总部的航空订票系统，并要查询或修改系统中记录所有班机当前订票数的数据库 B。设 B_i 为某班机的当前订票数，P_1 和 P_2 分别代表 2 个售

票处的售票进程，R_1 和 R_2 为进程执行时使用的工作寄存器。由于售票进程并发执行，且各自访问数据库 B 的时间是随机的，故有可能出现下面的访问序列（假定 B_i 的当前值为 x）：

P_1:R_1:=B_i;
R_1:=R_1+1
P_2:R_2=B_i;R_2:=R_2+1;
P_1:B_i:=R_1;P_2:B_i:=R_2

可见，B_i 的新值是 X+1，而不是 X+2。这里的 P_1 和 P_2 均为写者，显然，对于写者 B_i 为临界资源，因此写者应该互斥。

下面给出读者进程与写者进程的一般结构。

```
var mutex, wrt: semaphore;
readcount: integer;
begin
   seminit ;
readcount:=0
cobegin
procedure    reader;
begin
P(mutex);
Readcount:=readcount+1;
If    readcount=1 then    P (wrt);
V(mutex);
Reading is performing; P(mutex);
readcount:=readcount-1
if readcount=0 then V(wrt);
V (mutex);
End   Procedure   writer;
Begin
                P(wrt);
writing is performing;
V(wrt);
End
Coend
End;
```

结构化的同步/互斥机制——管程

建立管程的基本理由是：由于对临界区的执行分散在各进程中，这样不便于系统对临界资源的控制和管理，也很难发现和纠正分散在用户程序中的对同步原语的错误使用等问题。为此，应把分散的各同类临界区集中起来。并为每个可共享资源设立一个专门的管程来统一管理各进程对该资源的访问。这样既便于系统管理共享资源，又能保证互斥访问。管程主要由两部分组成：（1）局部于该管程的共享数据，这些数据表示了相应资源的状态。（2）局部于该管程的若干过程，每个过程完成关于上述数据的某种规定操作。为了实现对临界资源的互斥访问，管程每次只允许一个进程进入其内（即访问管程内的某个过程），这是由编译系统保证的。例：以环形缓冲池为例，给出环形缓冲池的管程结构

```
monitor ringbuffer;
```

```
var rbuffer：array[0. . n-1] of item；k, nextempty, nextfull: integer;
empty, full: condition;
procedure entry put (var product:item);
begin  if k=n wait (empty);
rbuffer [nextempty]: product;
k:=k+1；nextempty:=(nextempty+1) mod n;
signal (full);
end;
procedure entry get (var goods:item);
begin  if k =0 wait (full);
goods:=rbuffer [nextfull];
k:=k-1;
nextfull:=(nextfull+1) mod n;
signal (empty);
end;
begin
k:=0;
nextempty:=0;nextfull:=0;
end;
```

在利用管程解决生产者、消费者问题时，其中的生产者和消费者可描述为：

```
producer：begin
            repeat
                produce an item；
                 ringbuffer. put(item)；
            until    false；
            end
consumer：begin
              repeat
                ringbuffer. get (item)；
                  consume the item；
             end
```

进程的通信方式之二——消息缓冲

1． SEND(A)（发送消息）原语

发送消息原语被进程用于把消息发送到存放消息的缓冲区。A 是原语的参数，表示发送区的地址。其工作原理是：首先调用“寻找目标进程的 PCB”的程序查找接收进程的 PCB，如果接收进程存在，申请一个存放消息的缓冲区，消息缓冲区为空时，接收此消息的进程因等待此消息的到来而处于阻塞状态，则唤醒此进程，并把消息的内容、发送原语的进程名和消息等，复制到预先申请的存放消息的缓冲区，且将存放消息的缓冲区连接到接收进程的 PCB 上；如果接收进程不存在，则由系统给出一个“哑”回答；最后控制返回到发送消息的进程继续执行，或转入进程调度程序重新分配处理机。如果消息缓冲区已满，则返回到非同步错误处理程序入口进行特殊处理。如图 2-7 所示。

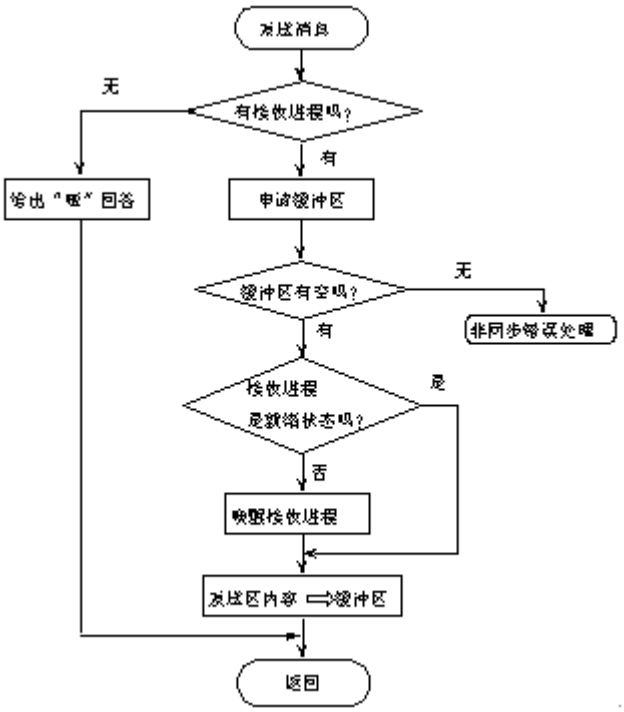


图 2-7 send 原语

2．READ（A）（读取消息）原语

READ(A)原语用来读取消息，接收进程读取消息之前，在自己的空间中确定一个接收区。当接收进程想要读取消息时，使用 READ(A)原语，A 是接收进程提供的接收区开始地址。如图 2-8 所示。

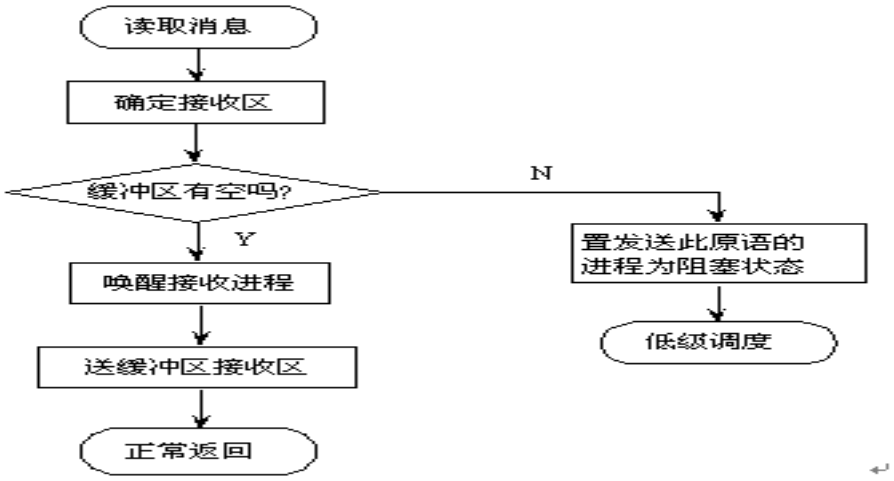


图 2-8 receive 原语

线程的基本概念

1．线程的引入

（1）创建进程。系统在创建进程时，必须为之分配其所必需的、除处理机以外的所有资源。如内存空间、I/O 设备以及建立相应的 PCB 结构。

（2）撤消进程。系统在撤消进程时，又必须先对这些资源进行回收操作，然后再撤消 PCB 结构。

（3）进程切换。在对进程进行切换时，由于要保留当前进程的 CPU 环境和设置新选中进程的 CPU 环境，为此需花费不少处理机时间。

2．线程与进程的比较

（1）调度：在传统的操作系统中，拥有资源的基本单位和独立调度、分派的基本单位都是进程。

（2）并发性：在引入线程的操作系统中，不仅进程之间可以并发执行，而且在一个进程中的多个

线程之间亦可并发执行，因而使操作系统具有更好的并发性，从而能更有效地使用系统资源和提高系统吞吐量。

（3）拥有资源：不论是传统的操作系统，还是设有线程的操作系统，进程都是拥有资源的一个独立单位，它可以拥有自己的资源。

（4）系统开销：由于在创建或撤消进程时，系统都要为之分配或回收资源，如内存空间、I/O 设备等。因此，操作系统所付出的开销将明显地大于在创建或撤消线程时的开销。

3．用户级线程和内核支持线程

比较两种线程的优缺点 ：

（1）线程的调度与切换速度：内核支持线程的调度和切换与进程的调度和切换十分相似。

（2）系统功能调用：当传统的用户进程调用一个系统功能调用时，要由用户态进入核心态，用户进程将被阻塞。当内核完成系统调用而返回时，才将该进程唤醒，继续执行。

（3）线程执行时间 ：对于只设置了用户级线程的系统，调度是以进程为单位进行的。在采用轮转调度算法时，各个进程轮流执行一个时间片，这对诸进程而言似乎是公平的。

三、本章小结：

进程可以理解为：“程序在并发环境中的执行过程。”它最基本的特征是并发性和动态性，一个进程至少应有三种状态，它们在一定条件下相互转化。

每一个进程都有惟一的进程控制块（PCB），它是进程存在的惟一标志。

PCB 表的物理组织方式有若干种，最常见的是线性方式、链接方式和索引方式。

UNIX 系统进程状态分为 10 种，在一定条件下进行变迁。

进程在活动过程中会彼此发生作用，主要是同步和互斥的关系。简单地说，同步是协作关系，互斥是竞争关系。

一次仅允许一个进程使用的资源称临界资源，对临界资源实施操作的那段程序称为临界区（CS）。利用信号量和 P，V 操作可以很容易地解决进程的互斥与同步问题。

本章可按下列主线掌握有关内容：

为了提高 CPU 的利用率——→提出多道程序设计——→为解决并行（并发）问题——→引入进程概念——→进程的特性（5 个）——→基本状态（3 种）——→状态的变化（4 种）——→进程的组成（3 部分）——→PCB 的组织方式（3 种）——→PCB 的管理（7 项）——→进程的相互作用——→进程管理命令等