第4章
进程及进程管理

- 进程的引入
- 进程概念
- 进程控制
- 进程的相互制约关系
- 进程同步机构
- 进程互斥与同步的实现

进程引入

1. 顺序程序及特点

(1) 计算

程序的一次执行过程称为一个计算，它由许多简单操作所组成。

(2) 程序的顺序执行

一个计算的若干操作必须按照严格的先后次序顺序地执行，这类计算过程就是程序的顺序执行过程。

(3) 顺序程序的特点

① 单道系统的工作情况

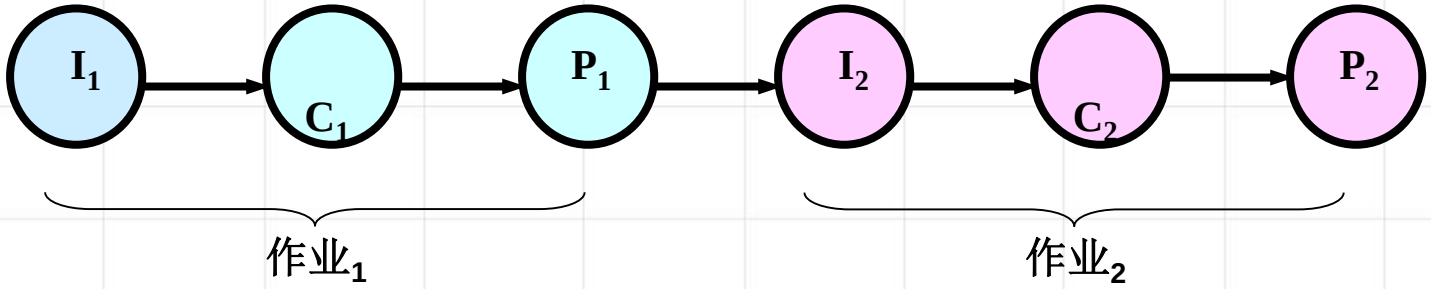
对用户作业的处理 ——

首先输入用户的程序和数据；然后进行计算；最后打印计算结果，即有三个顺序执行的操作。

I：输入操作

C：计算操作

P：输出操作



单用户系统中操作的先后次序图

② 顺序程序的特点

- **顺序性** —— 处理机的操作严格按照程序所规定的顺序执行。
- **封闭性** —— 程序一旦开始执行，其计算结果不受外界因素的影响。
- **可再现性** —— 程序执行的结果与它的执行速度无关 (即与时间无关)，而只与初始条件有关。

2. 并发程序

(1) 多道系统的工作情况

对n个用户作业的处理 ——

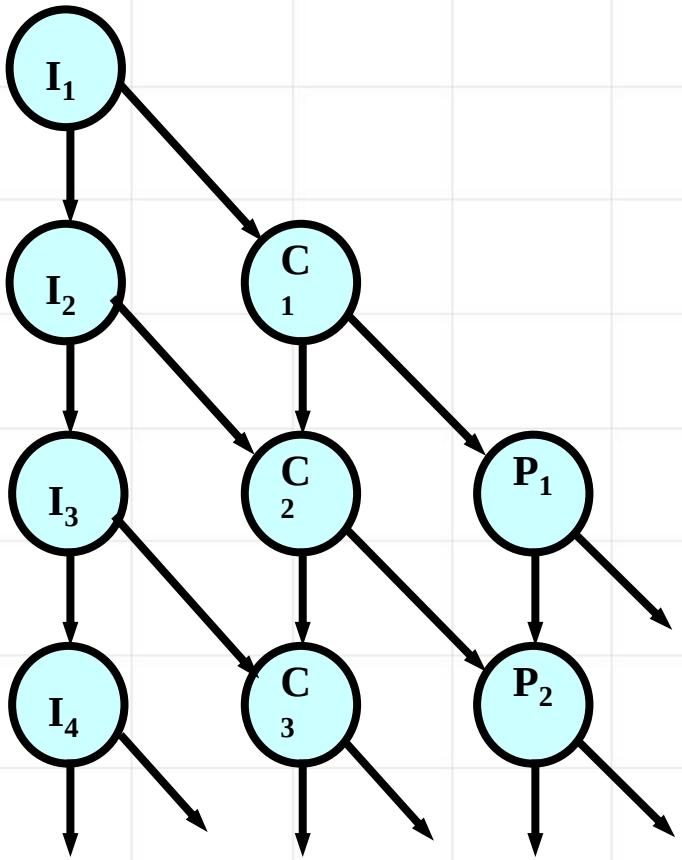
作业$_1$：　I_1　C_1　P_1

作业$_2$：　I_2　C_2　P_2

⋮　⋮　⋮　⋮

作业$_n$：　I_n　C_n　P_n

- 哪些程序段的执行必须是顺序的？为什么？
- 哪些程序段的执行是并行的？为什么？



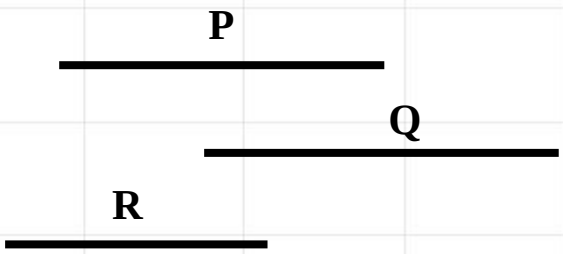
多用户系统中操作的先后次序图

(2) 什么是程序的并发执行

① 定义

若干个程序段同时在系统中运行，这些程序段的执行在时间上是重叠的，一个程序段的执行尚未结束，另一个程序段的执行已经开始，即使这种重叠是很小的一部分，也称这几个程序段是并发执行的。

② 三个并发执行的程序段



三个并发进程

③ 并行语句记号

cobegin

S_1； S_2； … ； S_n；

coend

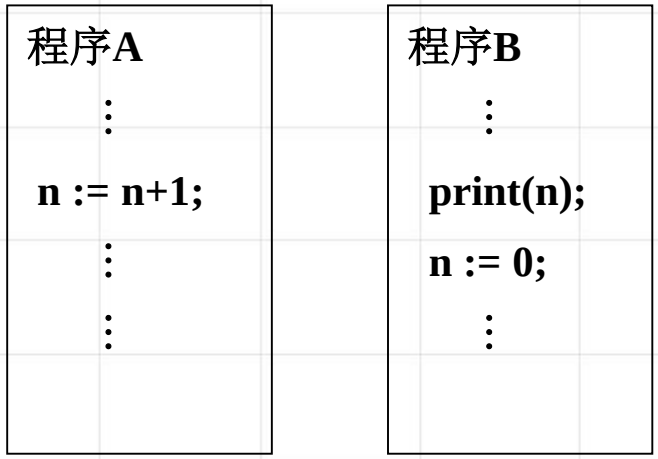
(3) 并发程序的特点

① 失去程序的封闭性和可再现性

若一个程序的执行可以改变另一个程序的变量，那么，后者的输出就可能有赖于各程序执行的相对速度，即失去了程序的封闭性特点。

i **例：**

讨论共享公共变量的两个程序，执行时可能产生的不同结果。程序A执行时对n做加1的操作；程序B打印出n值，并将它重新置为零。



共享变量的两个程序

ii 失去程序的封闭性和可再现性的讨论

程序A	程序B
⋮	⋮
n := n+1;	print(n);
⋮	n := 0;
⋮	⋮

共享变量的两个程序

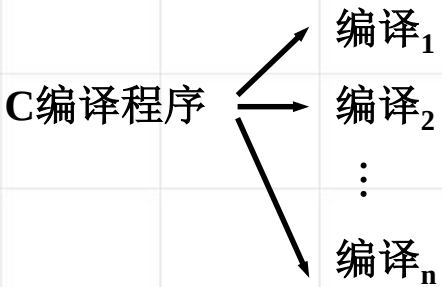
程序A的n :=n+1与程序B的两个语句的关系	之前	之中	之后
n的赋值	10	10	10
打印的结果	11	10	10
n的最终赋值	0	0	1

② 程序与计算不再一一对应

一个程序可以对应多个计算。

例1：

例2：



一个程序对应多个计算的例子

③ 程序并发执行的相互制约

- 间接的相互制约关系 —— 资源共享
- 直接的相互制约关系 —— 公共变量

3. 什么是与时间有关的错误

程序并发执行时，若共享了公共变量，其执行结果与各并发程序的相对速度有关，即给定相同的初始条件，若不加以控制，也可能得到不同的结果，此为与时间有关的错误。

进程概念

1. 进程定义

运行 ⟶ 暂停 ⟶ 运行

(1) 什么是进程

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

(2) 进程与程序的区别

① 程序是静态的概念，进程是动态的概念；

② 进程是一个独立运行的活动单位；

③ 进程是竞争系统资源的基本单位；

④ 一个程序可以对应多个进程，一个进程至少包含一个程序。

2. 进程的状态

(1) 进程的基本状态

① 运行状态(running)

该进程已获得运行所必需的资源，它的程序正在处理机上执行。

② 等待状态(wait)

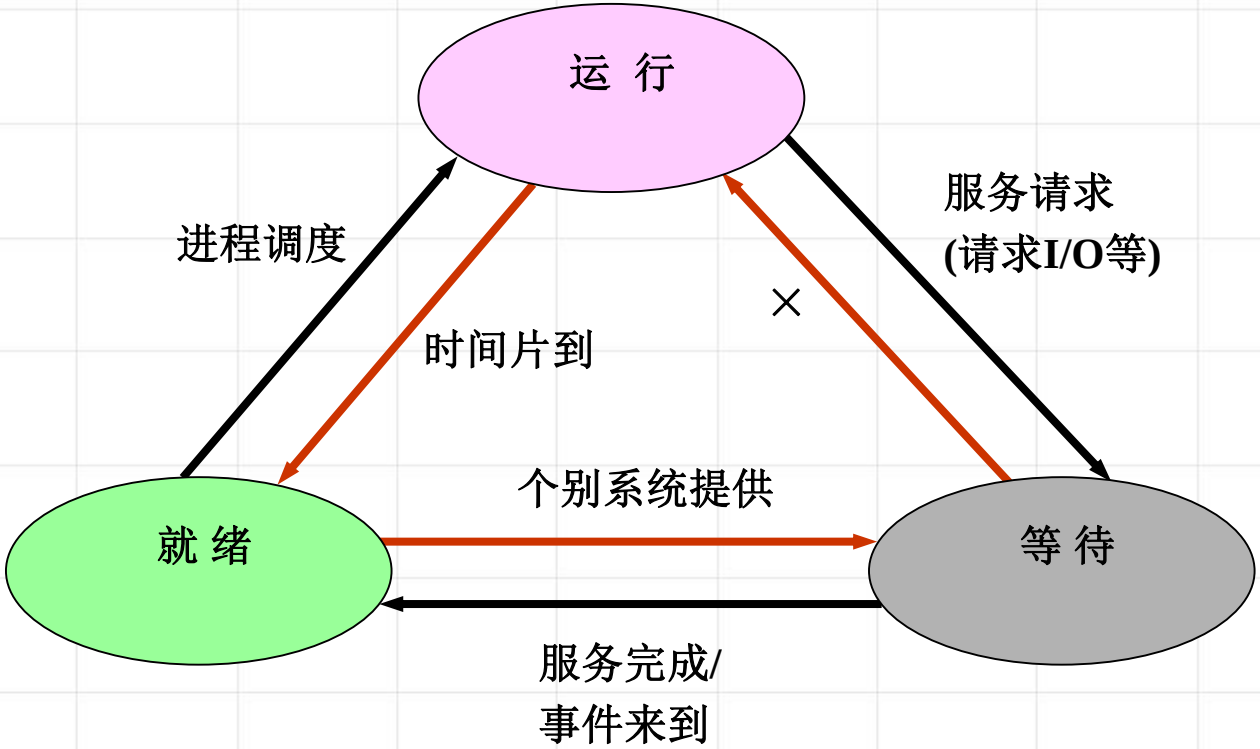
进程正等待着某一事件的发生而暂时停止执行。这时，即使给它CPU控制权，它也无法执行。

③ 就绪状态(ready)

进程已获得除CPU之外的运行所必需的资源，一旦得到CPU控制权，立即可以运行。

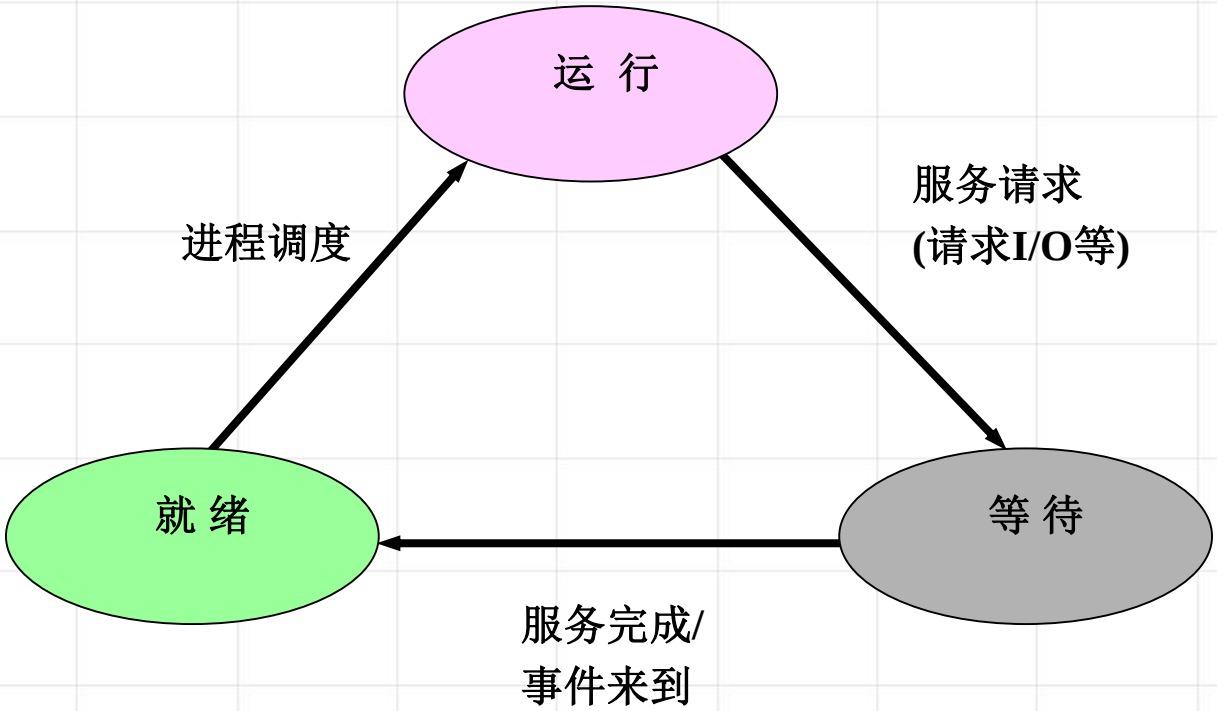
(2) 进程状态的变迁

① 进程状态可能的变迁



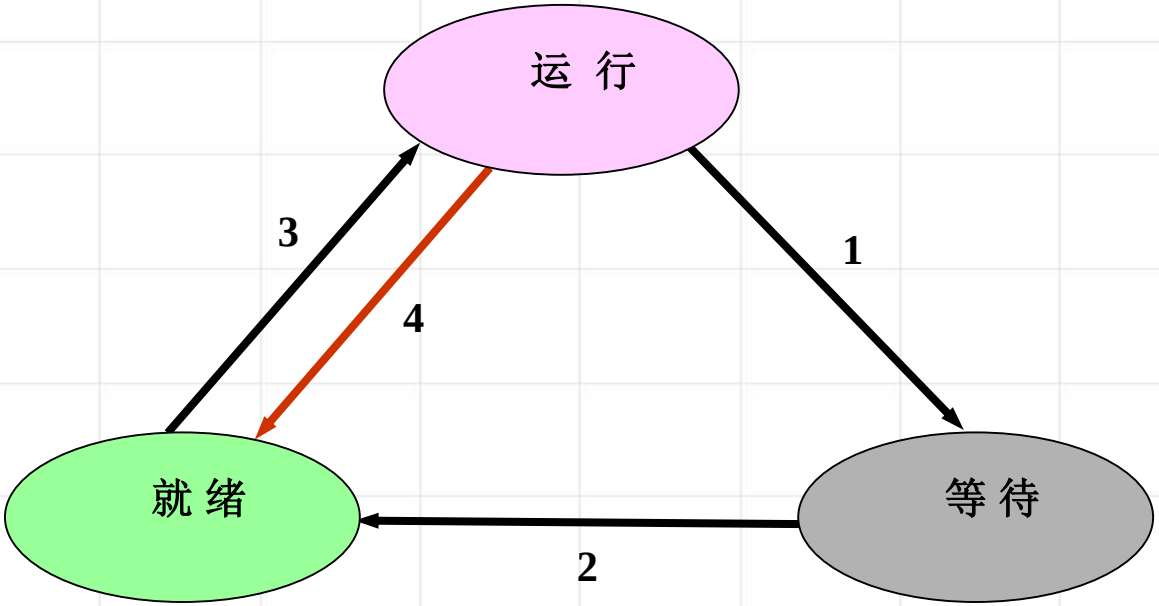
进程状态变迁图

② 具有进程基本状态的变迁图



进程状态变迁图

③ 讨论进程状态的变迁



进程状态变迁的讨论

变迁1——> 变迁3，是否会发生？需要什么条件？

变迁4——> 变迁3，是否会发生？需要什么条件？

(3) 讨论在多进程操作系统环境下程序的执行

① 例1：讨论3个排序程序在不同的操作系统环境中执行结果

程序A： 冒泡排序算法，在屏幕的左1/3处开设窗口显示其排序过程；

程序B： 堆排序算法，在屏幕的中1/3处开设窗口显示其排序过程；

程序C： 快速排序算法，在屏幕的右1/3处开设窗口显示其排序过程。

讨论在不支持多进程的操作系统下运行和在支持多进程的操作系统下运行的情况

i 在不支持多进程的操作系统下运行

依次运行程序A、程序B、程序C。

ii 在支持多进程的操作系统下运行

- 建立进程A、B、C；对应的程序分别是程序A、B、C；
- 若系统采用时间片轮转的调度策略，则在屏幕上有3个窗口，同时显示3个排序过程。

实际上这3个程序在轮流地占用CPU时间，由于CPU的高速度，使我们看到的是这3个程序在同时执行。

② 例2：讨论2个程序在不同的操作系统环境中执行结果

程序C：打印工资报表的程序；

程序D：计算1000以内所有素数并显示最后结果。

讨论在不支持多进程的操作系统下运行和在支持多进程的操作系统下运行。

i 在不支持多进程的操作系统下运行

依次运行程序C、程序D，可以看到，先是打印机不停地打印工资报表，打完后，接着运行程序C，不停地计算，最后显示所计算的结果。

ii 在支持多进程的操作系统下运行

- 建立进程C、D；对应的程序分别是程序C、D；
- 由于进程C是I/O量较大的进程，而进程D是计算量较大的进程，故在系统进程调度的控制下，两个进程并发执行。可以看到打印机不断打印工资报表；而处理机不停地计算，最后屏幕显示计算的结果。

3. 进程描述

(1) 什么是进程控制块

描述进程与其他进程、系统资源的关系以及进程在各个不同时期所处的状态的数据结构，称为进程控制块

PCB (process control block)。

(2) 进程的组成

进程组成的示意图

① 程序与数据

描述进程本身所应完成的功能

② PCB

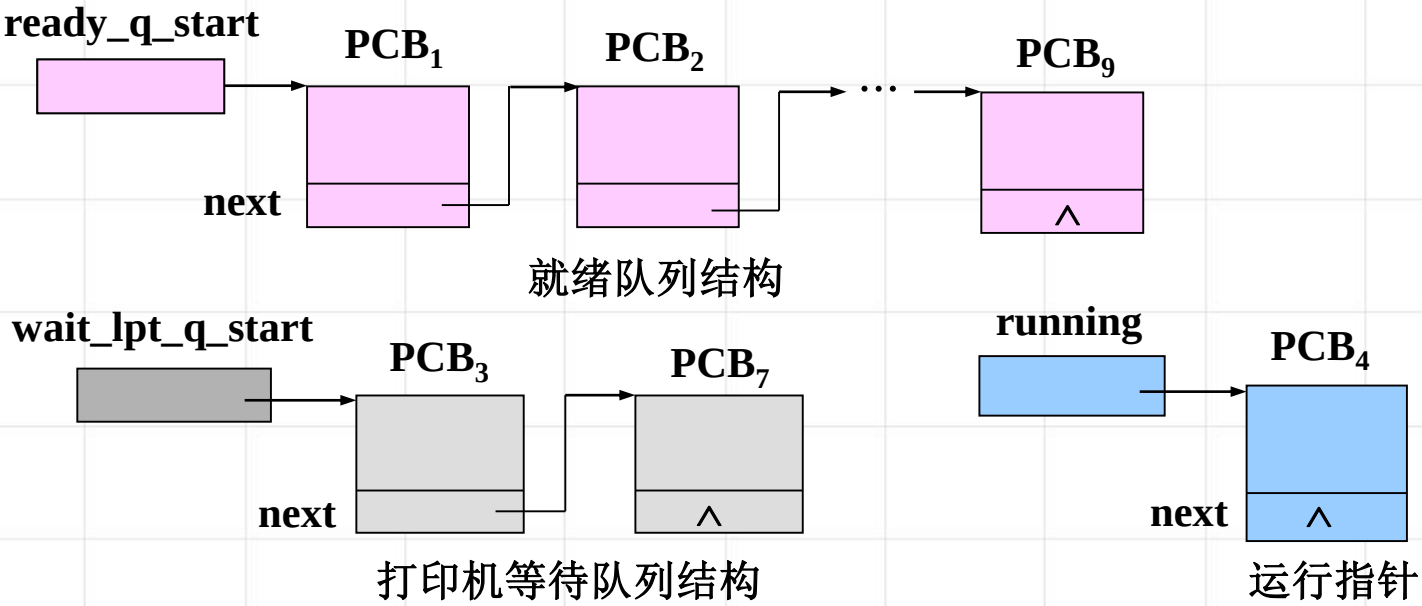
进程的动态特征，该进程与其他进程和系统资源的关系。

(3) 进程控制块的主要内容

① 进程标识符　　进程符号名或内部 id号

② 进程当前状态　　本进程目前处于何种状态

大量的进程如何组织？



进程队列结构示例

③ **当前队列指针next**

该项登记了处于同一状态的下一个进程的 PCB地址。

④ **进程优先级**

反映了进程要求CPU的紧迫程度。

⑤ **CPU现场保护区**

当进程由于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。

⑥ **通信信息**

进程间进行通信时所记录的有关信息。

⑦ **家族联系**

指明本进程与家族的联系

⑧ **占有资源清单**

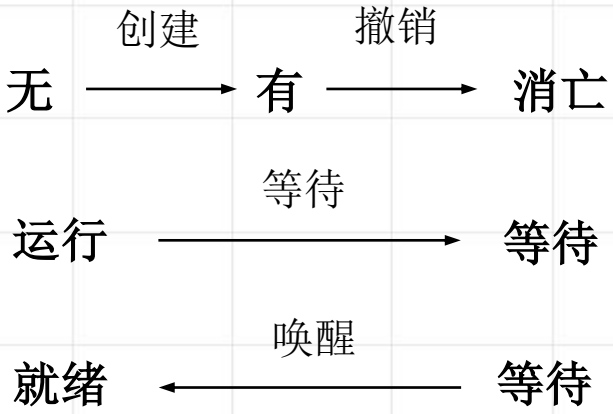
进程控制

1. 进程控制的概念

(1) 进程控制的职责

对系统中的进程实施有效的管理，负责进程状态的改变。

① 进程状态变化



② 常用的进程控制原语

创建原语、撤消原语、阻塞原语、唤醒原语

(2) 进程创建

① 进程创建原语的形式

create (name，priority)

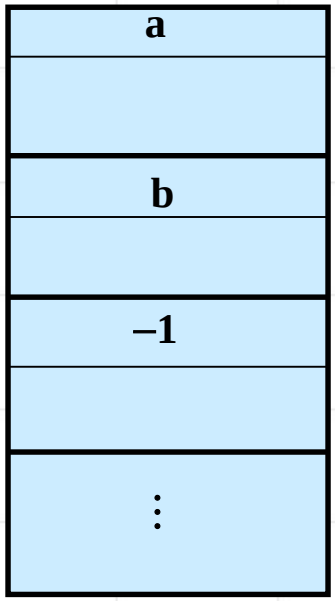
- name为被创建进程的标识符
- priority为进程优先级

② 进程创建原语的功能

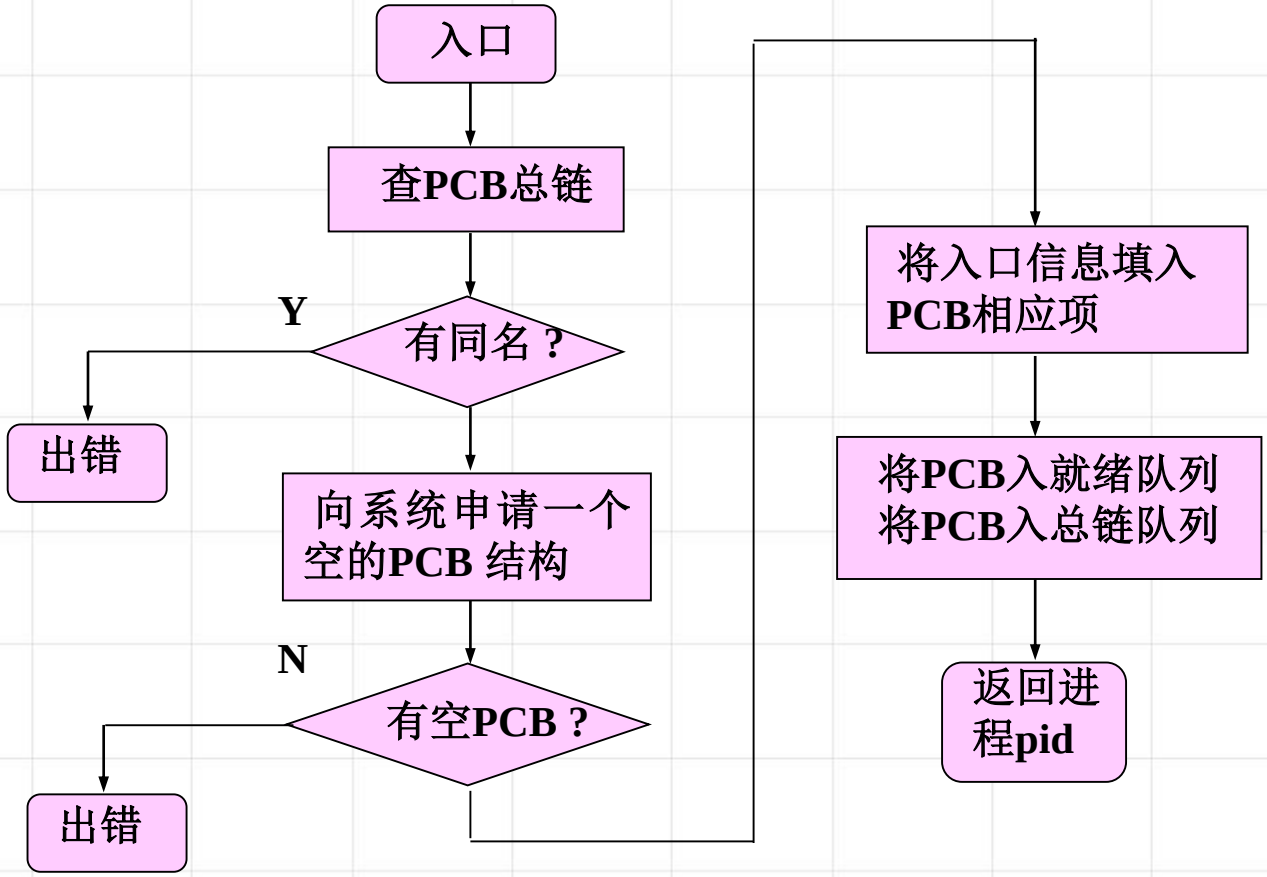
创建一个具有指定标识符的进程，建立进程的PCB结构。

③ 进程创建原语的实现

- PCB池
- 进程创建原语的实现框图



PCB池示意图



进程创建原语流程图

(3) 进程撤销

① 进程撤销原语的形式

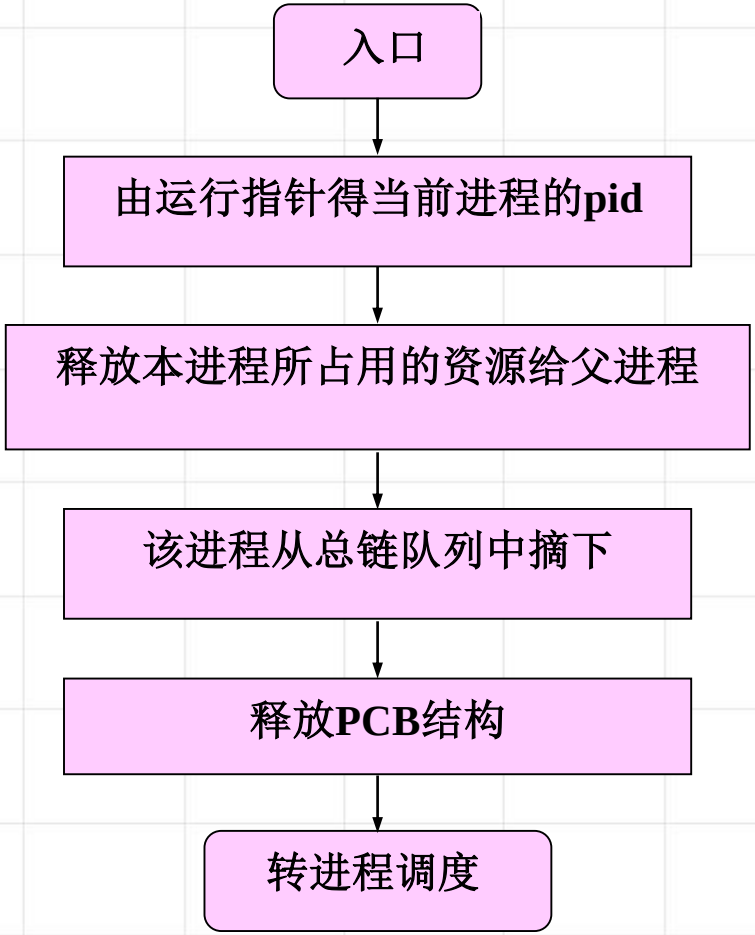
当进程完成任务后希望终止自己时使用进程撤消原语。

Kill (或exit)

② 进程撤销原语的功能

撤消当前运行的进程。将该进程的PCB结构归还到PCB资源池，所占用的资源归还给父进程，从总链队列中摘除它，然后转进程调度程序。

③ 进程撤销原语的实现



进程撤销原语流程图

(4) 进程等待

① 进程等待原语的形式

当进程需要等待某一事件完成时，它可以调用等待原语挂起自己。

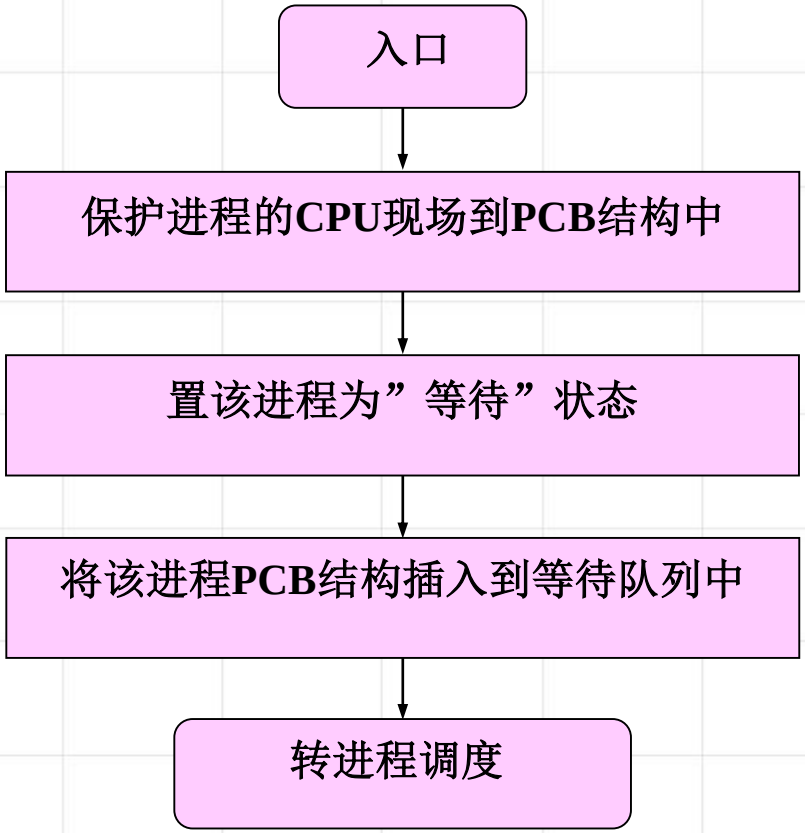
susp(chan)

入口参数chan：进程等待的原因

② 进程等待原语的功能

中止调用进程的执行，并加入到等待chan的等待队列中；最后使控制转向进程调度。

③ 进程等待原语的实现



进程等待原语流程图

(5) 进程唤醒

① 进程唤醒原语的形式

当处于等待状态的进程所期待的事件来到时，由发现者进程使用唤醒原语叫唤醒它。

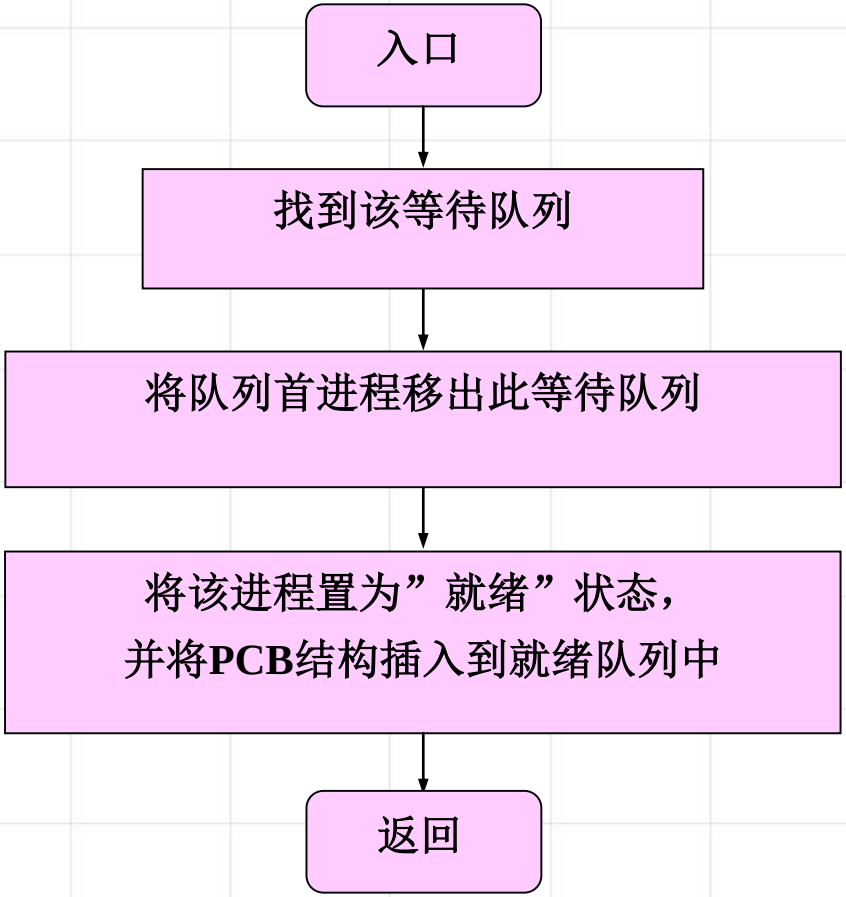
wakeup(chan)

入口参数chan：进程等待的原因。

② 进程唤醒原语的功能

当进程等待的事件发生时，唤醒等待该事件的进程。

③ 进程唤醒原语的实现



进程唤醒原语流程图

进程之间的约束关系

1. 进程互斥的概念

(1) 临界资源

① 例1：两个进程A、B共享一台打印机

设：x代表某航班机座号，p_1和p_2两个售票进程，售票工作是对变量x加1。这两个进程在一个处理机C上并发执行，分别具有内部寄存器r_1和r_2。

② 例2：两个进程共享一个变量x

两个进程共享一个变量x时，两种可能的执行次序：

- **A：**

p_1： $r_1 := x$； $r_1 := r_1+1$； $x := r_1$ ；

p_2： $r_2 := x$； $r_2 := r_2+1$； $x := r_2$ ；

- **B：**

p_1： $r_1 := x$； $r_1 := r_1+1$； $x := r_1$ ；

p_2： $r_2 := x$； $r_2 := r_2+1$； $x := r_2$ ；

- 设x的初值为10，两种情况下的执行结果：

情况A： x = 10+2

情况B： x = 10+1

③ 临界资源的定义

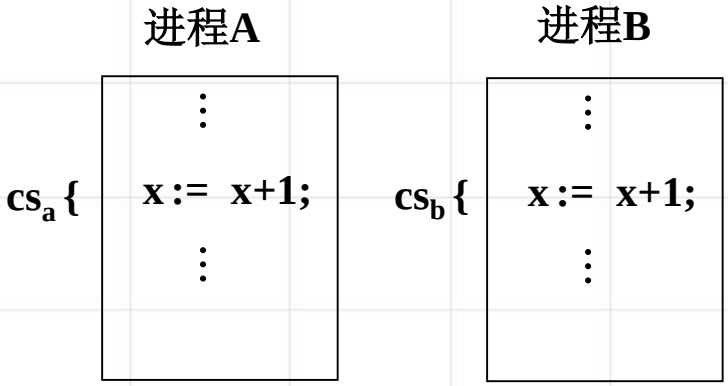
一次仅允许一个进程使用的资源称为临界资源。

硬件：如输入机、打印机、磁带机等

软件：如公用变量、数据、表格、队列等

(2) 临界区

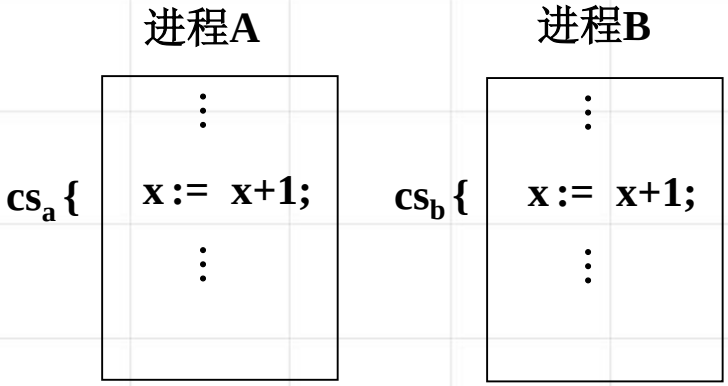
临界区是进程中对公共变量 (或存储区)进行审查与修改的程序段，称为相对于该公共变量的临界区。



进程临界区示意图

(3) 互斥

在操作系统中，当某一进程正在访问某一存储区域时，就不允许其他进程来读出或者修改存储区的内容，否则，就会发生后果无法估计的错误。进程间的这种相互制约关系称为互斥。



进程临界区示意图

2. 进程同步的概念

(1) 什么是进程同步

并发进程在一些关键点上可能需要互相等待与互通消息，这种相互制约的等待与互通消息称为进程同步。

(2) 进程同步的例

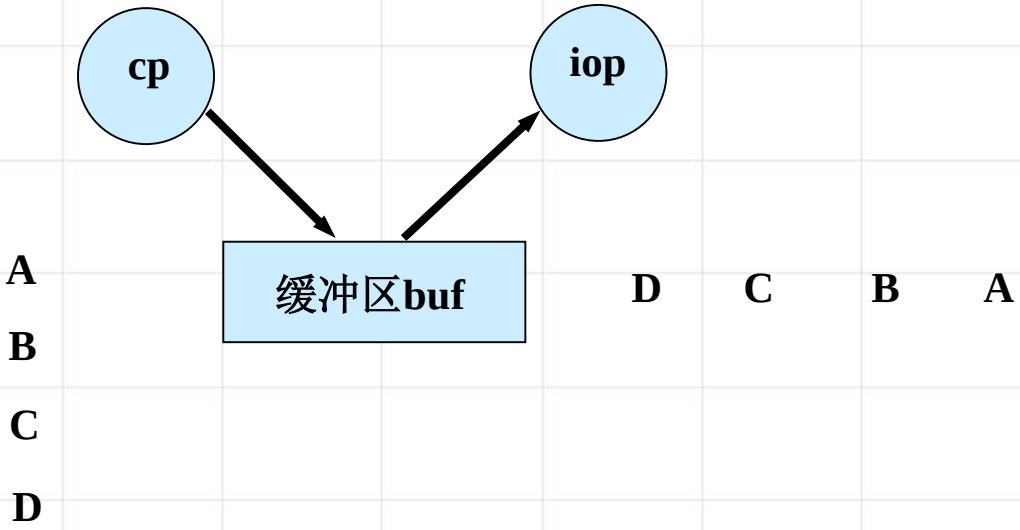
① 病员就诊

看病活动：	化验活动：
⋮	⋮
要病人去化验；	需要进行化验？
⋮	⋮
等化验结果；	进行化验；
⋮	开出化验结果；
继续诊病；	⋮

进程同步活动示意图

② 共享缓冲区的计算进程与打印进程的同步

计算进程 cp和打印进程 iop公用一个单缓冲



两个进程共享一个缓冲区示意图

进程同步机构

1. 锁和上锁、开锁操作

(1) 什么是锁

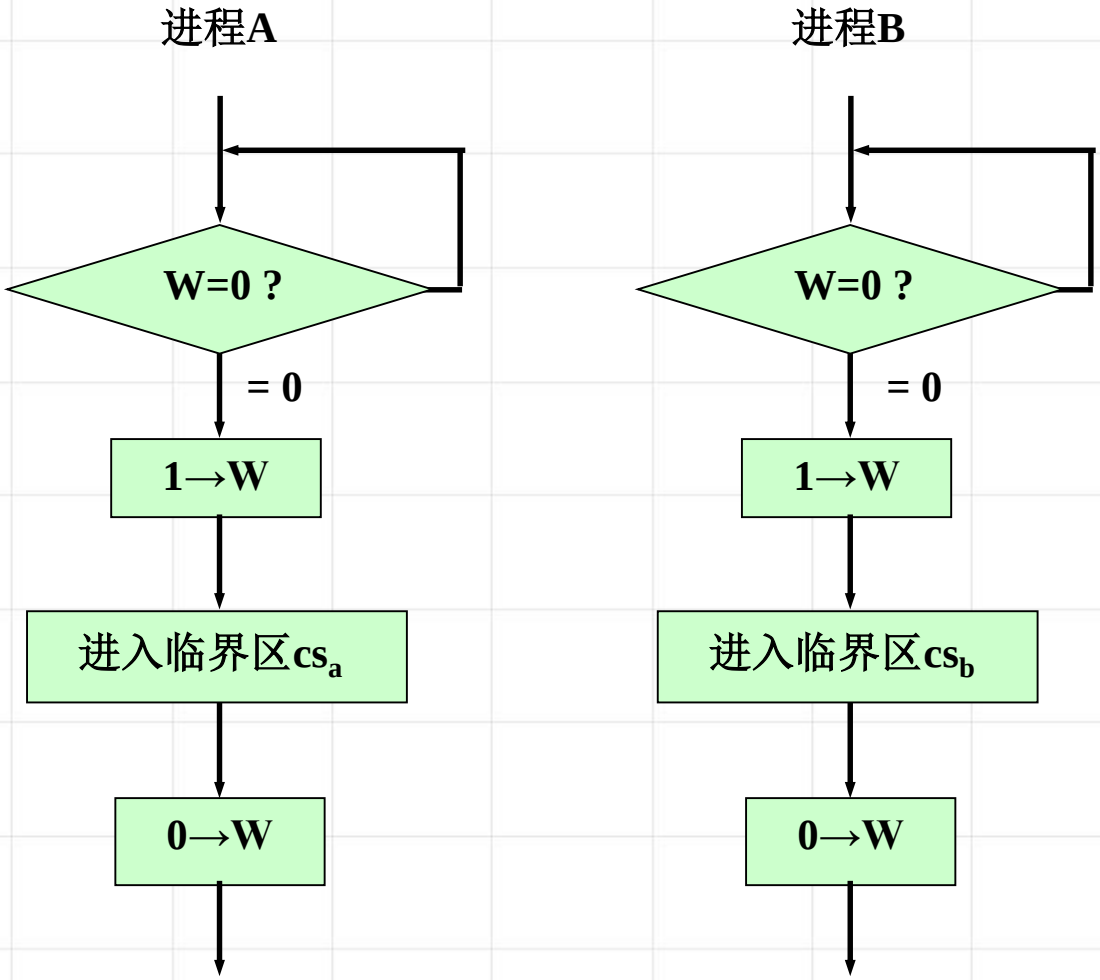
用变量w代表某种资源的状态，w称为“锁”。

(2) 上锁操作和开锁操作

- 检测w的值 (是0还是1);
- 如果w的值为1，继续检测;
- 如果w的值为0，将锁位置1 (表示占用资源)，进入临界区执行。 **(此为上锁操作)**

- 临界资源使用完毕，将锁位置0。 **(此为开锁操作)**

(3) 进程使用临界资源的操作



两个进程使用临界资源的操作

(4) 上锁原语和开锁原语

① 上锁原语

```
算法  lock
输入：锁变量w
输出：无
{
    test：   if (w为1)
    goto  test;
                /* 测试锁位的值   */
    else   w=1;      /*   上锁   */
}
```

② 开锁原语

```
算法  unlock
输入：锁变量w
输出：无
{
    w=0；/*开锁*/
}
```

2. 信号灯和P、V操作

(1) 什么是信号灯

信号灯是一个确定的二元组 (s，q)，s是一个具有非负初值的整型变量，q是一个初始状态为空的队列。操作系统利用信号灯的状态对并发进程和共享资源进行控制和管理。

信号灯是整型变量。

变量值 ≥ 0 时，表示绿灯，进程执行；

变量值 ＜ 0 时，表示红灯，进程停止执行。

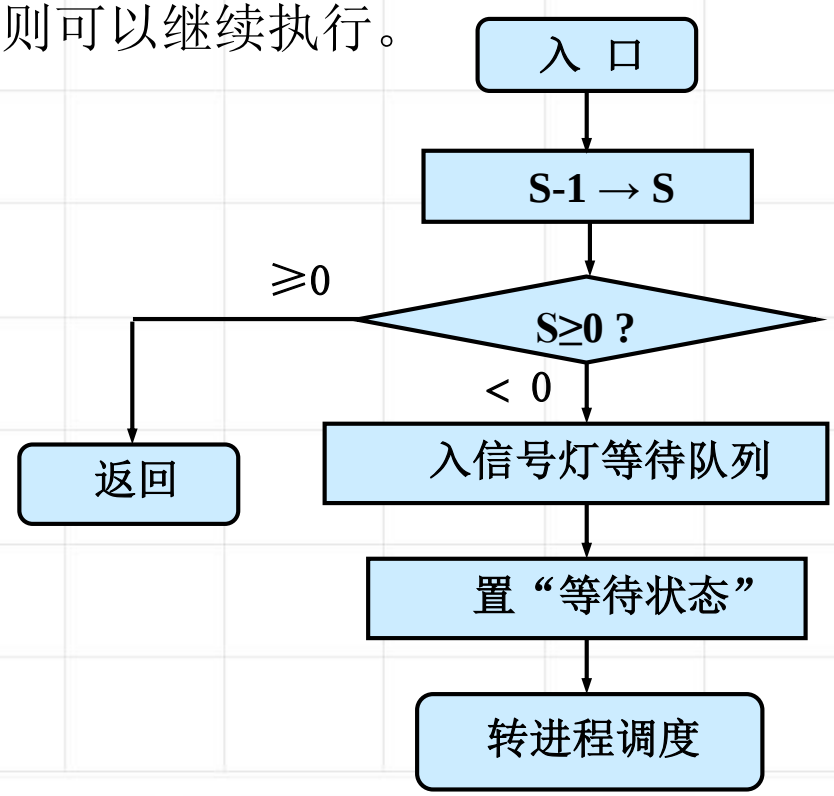
注意：创建信号灯时，应准确说明信号灯 s 的意义和初值(这个初值绝不能为负值)。

(2) P 操作

① P 操作的定义

对信号灯s的 p操作记为 p(s)。p(s)是一个不可分割的原语操作，即取信号灯值减1，若相减结果为负，则调用p(s)的进程被阻，并插入到该信号灯的等待队列中，否则可以继续执行。

② P 操作的实现



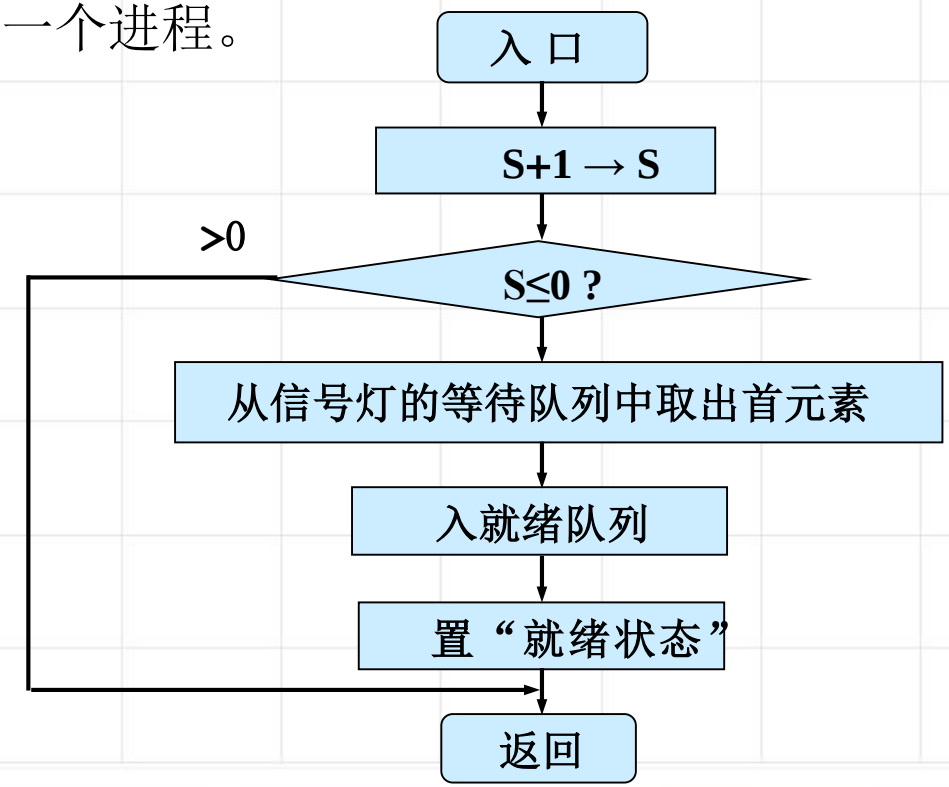
P 操作原语流程图

(3) V 操作

① V 操作的定义

对信号灯s的 v操作记为 v(s)。v(s)是一个不可分割的原语操作，即取信号灯值加1，若相加结果大于零，进程继续执行，否则，要帮助唤醒在信号灯等待队列上的一个进程。

② V 操作的实现



V 操作原语流程图