第二章 进程管理

2.1 进程及其状态

2.2 进程控制块

2.3 进程控制

2.4 线程

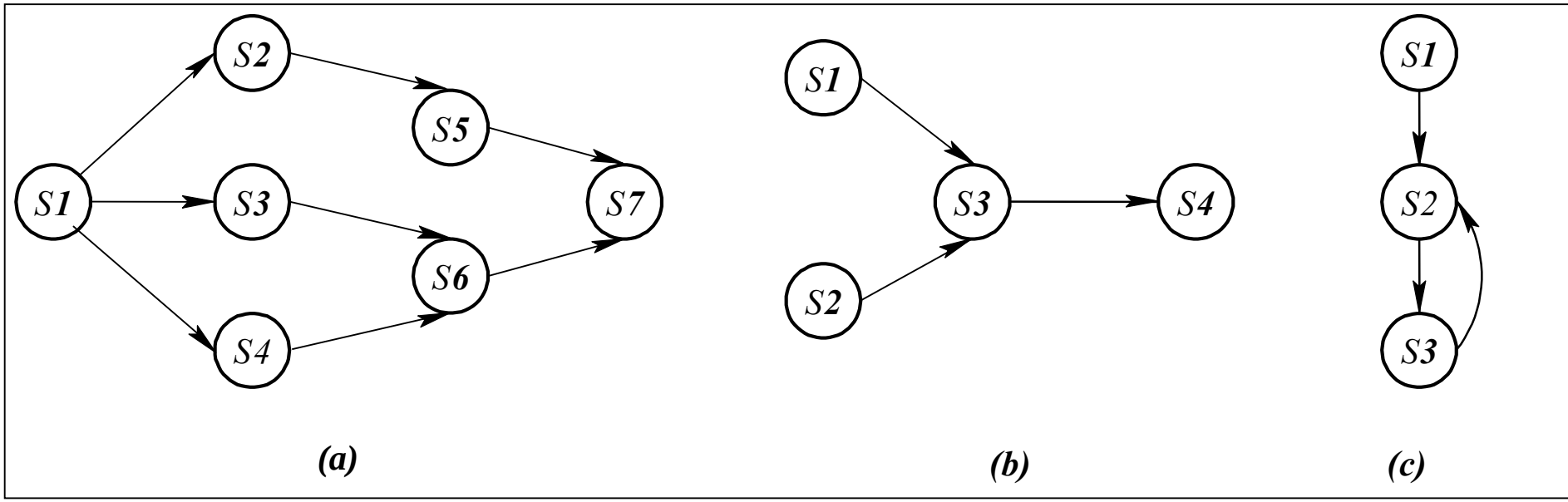
2.1 进程及其状态

2.1.1 进程的引入

前趋图(Procedence Graph)

前趋图是一个有向无循环图DAG(Directed Acyclic Graph)。

在前趋图中，每个结点代表一个事件。在操作系统中，可以将结点看作语句、程序段或进程等。结点间的有向边表示两个结点间的前趋关系，记为“→”。若有S1→S2，则表示只有当S1完成之后，S2才可以执行。称S1是S2的直接前趋，S2是S1的直接后继。



前趋图

注意：前趋图中必须不存在循环！

2.1.1 进程的引入

1. 程序的顺序执行

所谓程序的顺序执行，是指一个程序运行时独占整个系统资源，处理机严格按照程序所规定的顺序进行操作。

在一个程序段中，只有前面的一个操作执行完，才能进行后面一个操作。

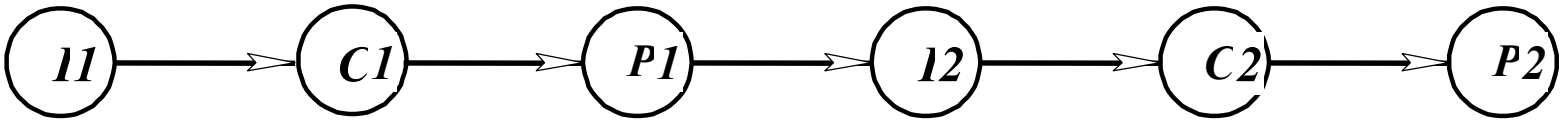
例如下述程序段：

- **S1: a=x+y；**
- **S2: b=a-5；**
- **S3: c=b+1；**

一个程序段中语句的顺序执行

2.1.1 进程的引入

在一个系统中，若要执行几个程序段，只有当一个程序段执行完后才能执行另一个程序段中的操作。例如：一个程序段若由输入(I)、计算(C)、输出(P)等操作组成，两个程序段的顺序执行则如图所示。



两个程序段的顺序执行

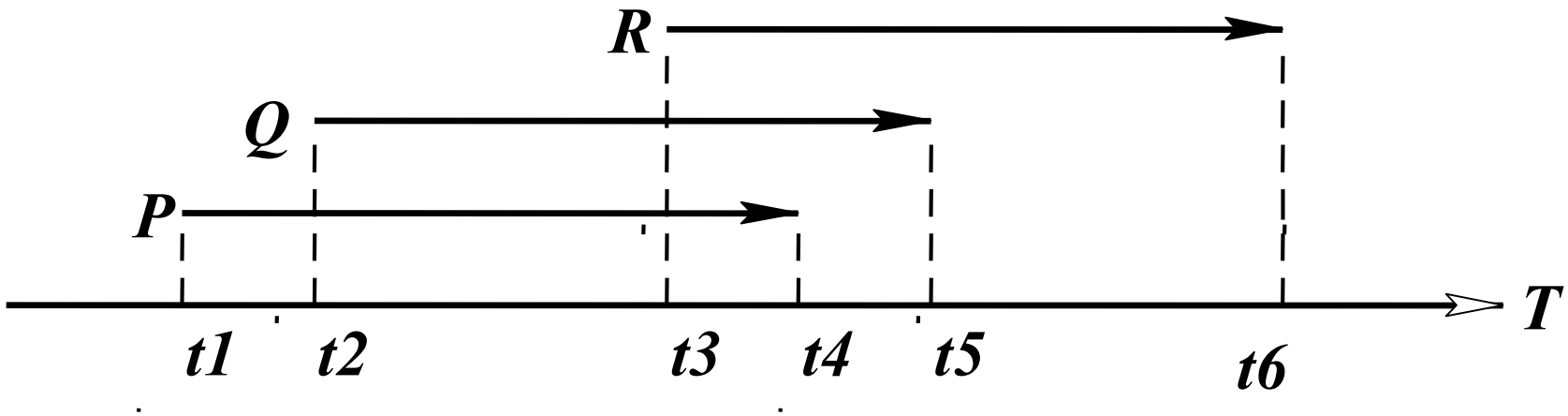
2.1.1 进程的引入

2. 程序顺序执行的特征

- **顺序性：** 处理机的操作严格按照程序所规定的操作顺序执行，时间上完全有序，即只有前一个操作执行完以后，才能进行后继操作。
- **运行环境的封闭性：** 程序执行时独占系统资源，系统内各种资源的状态（初始状态除外）只能被本程序所改变，其执行结果不受外界因素干扰。
- **程序运行结果可再现性：** 只要程序执行的环境与初始状态不变，当重复执行时，所获得的结果相同，与执行速度无关。

3. 程序的并发执行

- 并行操作指多个操作在计算机系统的各个硬件部件上同时进行。这是真正意义上的“同时”操作，称之为并行操作。
- 所谓并发执行，是指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。例如，在一个时间段内，一个CPU在为多道程序工作，而在某一个瞬间，一个CPU只能运行一道程序，它只是在多道程序中快速切换，给人以CPU“同时”运行几道程序的感觉。每个程序内部仍是按顺序执行，但是多个程序的执行过程是可以交叉的，这是一种伪并行。



4. 程序并发执行的特征

①间断性：多个程序段并发执行时，由于共享资源或由于相互合作而形成执行时的相互制约关系，使得每个程序段执行时产生了间断性。将导致程序具有“执行→暂停→执行”这种间断的活动规律。

4. 程序并发执行的特征

② **失去封闭性：**多个程序段并发执行时，每个程序段不再独占系统资源，执行时受外界因素影响。例如，当一个用户的程序段执行中使用某个I/O设备时，其他用户的程序段申请使用该设备，就必须等待。

③ **不可再现性：**多个程序段并发执行时，产生了非封闭性，不再独占系统资源，此时，即使程序执行的环境与初始状态不变，重复执行时运算结果通常也不可再现。

example

- 例如:两个并发执行的程序段A与B共享一个变量N，两个程序段推进速度发生变化时，可能导致程序运行结果不确定。假定N初始值为3。

A:	B:
⋮	⋮
N=N+1；	PRINT N；
⋮	N=0；
	⋮

程序的执行结果与相对执行速度有关，有时，程序的多次执行会有不同的结果。

程序 *不一一对应* ⟷ 执行结果

进程：程序的一次执行过程。

2.1.2 进程的概念

1. 进程定义

1961年，进程的概念首先由美国麻省理工学院在MULTICS系统中引入。随后，许多人都对进程下过定义，如：

①一个正在执行中的程序。

②一个正在计算机上执行的程序实例。

③能分配给处理机并由处理机执行的实体。

④一个具有以下特征的活动单元：一组指令序列的执行、一个当前状态和相关的系统资源集合。

⑤程序在一个数据集合上运行的过程，是系统进行资源分配和调度的一个独立单位。

根据1978年在庐山召开的全国操作系统会议上的讨论，认为“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”。

传统OS中进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位。

2.1.2 进程的概念

2. 进程的特征

- 结构特征 ☞
- 动态性 ☞
- 并发性 ☞
- 独立性
- 异步性

1) 结构特征

- 进程实体：

三部分组成：

- **Program code** 程序代码 —功能实现
- **Set of data** 程序段操作的数据集 —工作区
- 进程控制块(PCB)：系统感知进程的唯一标识

不同的PCB代表不同的进程

- PCB中记录了进程在系统中的每一次活动和变化情况。操作系统根据PCB中各项的变化，掌握进程所处的状态，达到控制进程的目的。

2) 动态性

- 动态性是进程的最基本的特征。表现在：由创建而产生，由调度而执行，由撤消而消亡。
- 进程在系统中有一定的生命周期。

- 而程序仅是存放于某种介质的有序指令的集合。
- 程序是静态的。

3)并发性、独立性、异步性

- 并发性
  - 引入进程的目的是为了程序的并发执行。
  - 没有创建进程的程序是不能并发执行的。
- 独立性
  - 进程是系统进行资源分配和调度的独立单位。
- 异步性
  - 进程按各自独立的、不可预知的速度向前推进。

2.1.3 进程的状态模型

1. 两状态模型

进程的两状态：运行态、非运行态。

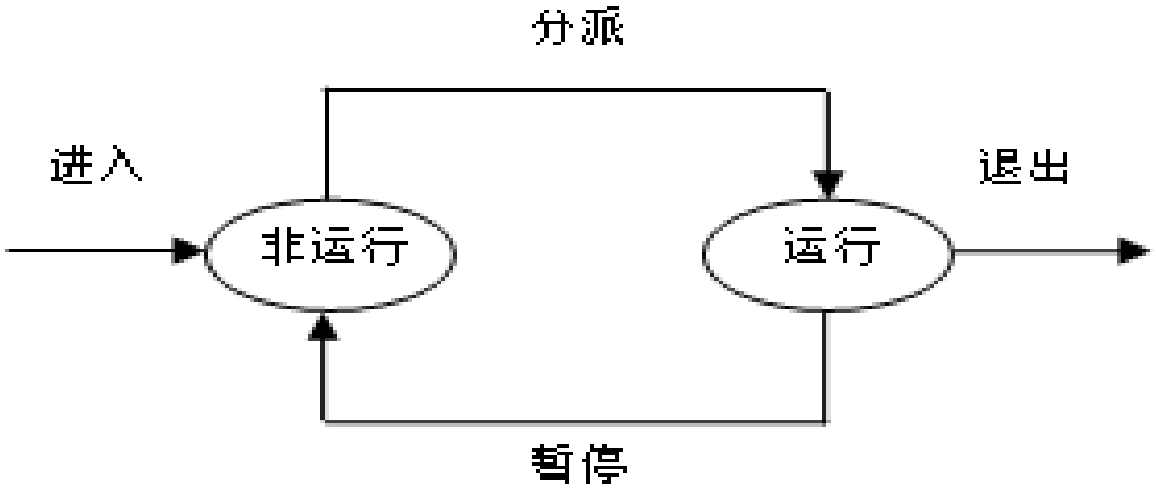


图 2-3　两状态模型

2. 三状态模型

进程的三个基本状态：就绪状态、执行状态与阻塞状态。进程在运行过程中必处于这三个基本状态之一。

- 就绪状态：进程获得必要资源，例如内存等，已经具备了执行条件，只是没有空闲的CPU，处于等待CPU的状态。在系统中，将处于就绪状态的多个进程的PCB排成就绪队列。

- 运行状态：进程已经获得必要的资源及CPU，它的程序正在执行中。在多处理机系统中，可以有多个进程处于执行状态。在单处理机系统中，只能有一个进程处于执行状态。
- 阻塞状态：进程因某等待事件发生（例如**I/O**请求、某些原语操作等）而处于暂停执行的状态，此时即使将**CPU**分配给它，也无法执行。在系统中，将处于阻塞状态的进程的**PCB**排成一个阻塞队列，或因阻塞原因不同而排在不同队列中。

2. 三状态模型

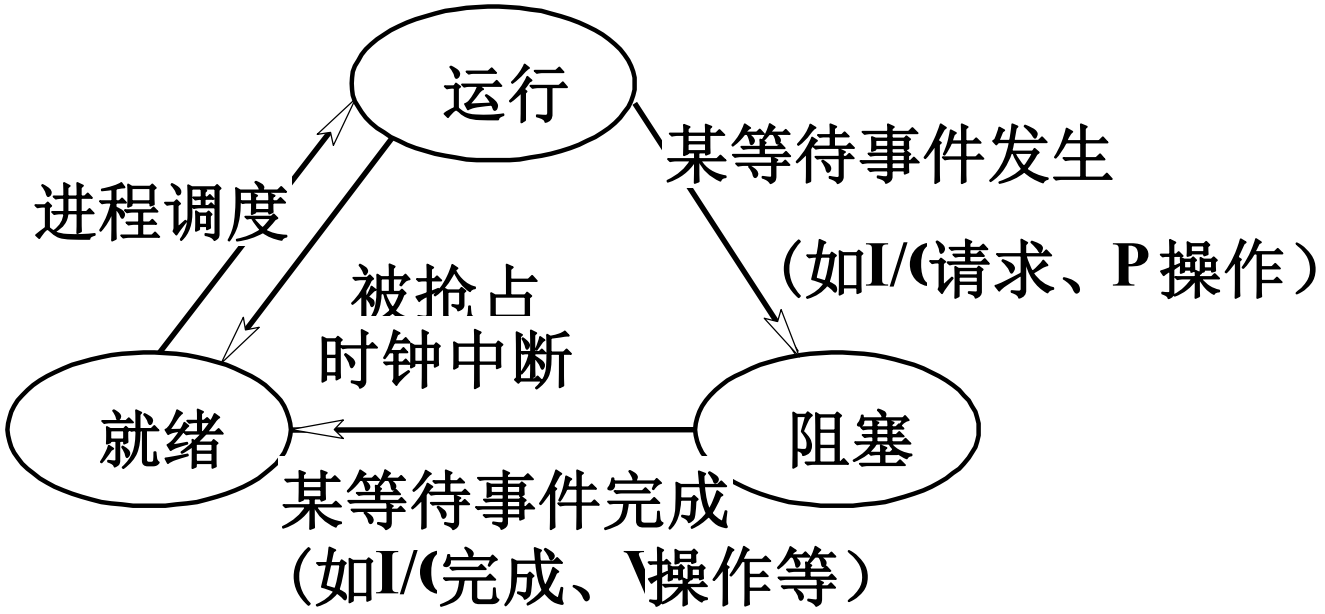


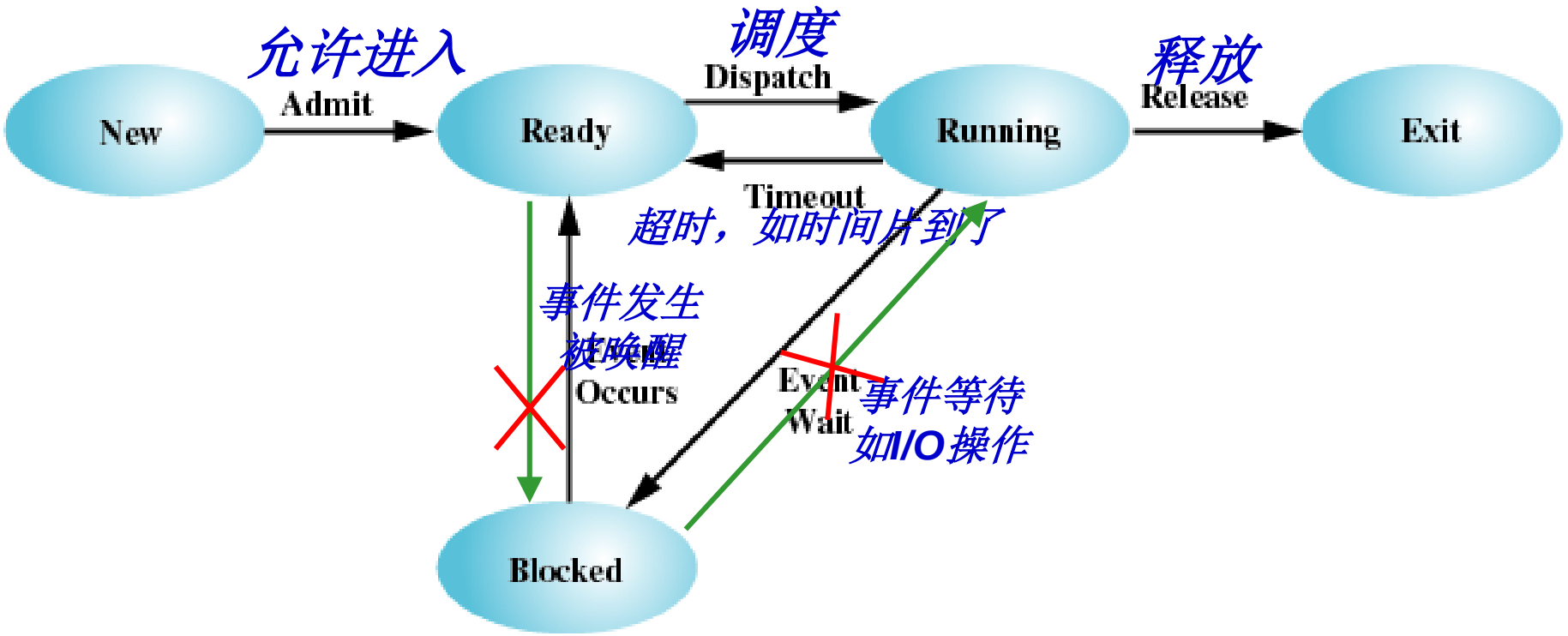
图2-5 三状态进程模型

3. 五状态模型

在多道程序环境中，只有作为进程时才能在系统中运行。因此，为使程序能运行，就必须为它创建进程。操作系统在定义一个新进程时，要做一系列相关的工作。也就是说并不是所有的进程一创建就马上可以运行的。

为便于进程管理，有必要增加一种有用的状态，称为新建态，对应于刚刚创建的进程，操作系统还没有把它加入到就绪队列中，通常是进程控制块已经创建但还没有加载到内存中。同样，进程从系统中退出时，也增加一种有用的状态，进程被终止直到释放PCB所处的状态，我们称为退出态。

3. 五状态模型



- 注意：不存在以下状态转换
  - 阻塞——执行　　就绪——阻塞

4. 挂起状态的引入

1. 终端用户的需要：发现可疑问题
2. 父进程的需要：考查、修改子进程、协调子进程间活动
3. 负荷调节的需要：调整负荷，使系统正常运行。
4. 操作系统的需要：检查资源使用情况
5. 对换的需要：缓和内存紧张

挂起操作（Suspend）：

- 对正在执行的进程，则停止执行。
- 对就绪状态的进程，则停止调度，并回收资源。
- 对阻塞状态的进程，则停止条件的发生。

5. 引入挂起后进程状态的转换

活动就绪→静止就绪： 当进程处于未被挂起的就绪状态时，称活动就绪状态，此时进程可接受调度。当它被挂起时，便转变为静止就绪状态，此时不再接受调度。通常，新建进程进入活动就绪状态，以便尽快得到调度。

活动阻塞→静止阻塞： 当进程处于未被挂起的阻塞状态时，称活动阻塞状态，此时进程可因某等待事件的完成而转变为活动就绪状态。当处于活动阻塞状态的进程被挂起时，便转变为静止阻塞状态。

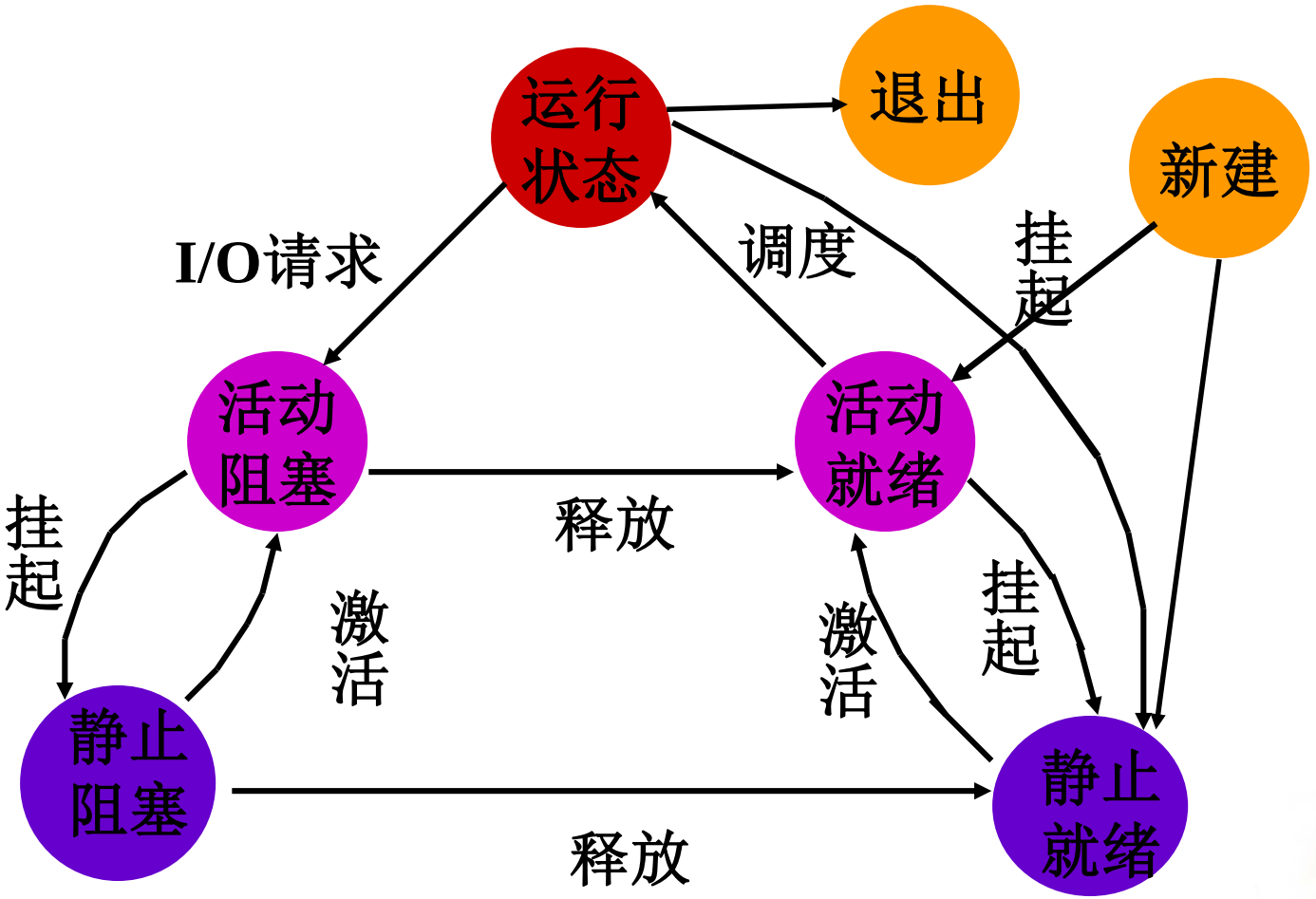
静止阻塞→静止就绪：处于静止阻塞状态的进程在所期待的事件出现后，将转变为静止就绪状态，仍处于静止状态。

静止就绪→活动就绪：处于静止就绪状态的进程被激活后，转变为活动就绪状态，进程可重新接受调度。

静止阻塞→活动阻塞：处于静止阻塞状态的进程被激活后，转变为活动阻塞状态，当所期待的事件完成后，将转变为活动就绪状态。

运行→静止就绪：处于执行状态的进程被挂起后处于静止就绪状态，暂停执行，不再接受调度。

6. 七状态模型



2.2 进程控制块

2.2.1 进程控制块的作用

- 为描述进程动态变化过程以及对进程进行控制与管理而设的数据结构。进程与PCB一一对应，操作系统根据PCB了解进程的存在，是进程是否存在的唯一标志。
- PCB常驻内存，属于系统空间，只有操作系统程序才能访问，用户程序不得访问。

2.2.1 进程控制块的作用

- 作为独立运行基本单位的标志。
- 能实现间断性运行方式。
- 提供进程管理所需要的信息。
- 提供进程调度所需要的信息。
- 实现与其它进程的同步与通信。

2.2.2 进程控制块的内容

进程标识信息

进程标识符：

- ✓ 外部标识符：创建者提供，字母、数字组成，便于记忆。
- ✓ 内部标识符：唯一数字标识符，一个进程的序号，方便系统使用。

父进程标识符：创建本进程的进程的标识符

用户标识符：进程所属用户的标识符

2.2.2 进程控制块的内容

- 处理机状态信息：中断点现场信息

✓ 通用寄存器：用户可视寄存器，可被用户程序访问，暂存信息。

✓ 程序计数器：存放要访问的下一条指令的地址。

✓ 程序状态字：含状态信息，如条件码、执行方式、中断屏蔽标志等。

✓ 用户栈指针：存放过程和系统调用参数及调用地址。

2.2.2 进程控制块的内容

- 进程调度信息：
  - ✓进程优先级
  - ✓进程状态
  - ✓进程调度所需的其它信息：进程已等待CPU的时间总和、进程已执行的时间总和等。

2.2.2 进程控制块的内容

- 进程控制信息：

✓ 程序和数据的地址：该进程的程序和数据所在的内存或外存地址。

✓ 资源清单：除CPU外，所需和已经分配到的资源。

✓ 进程同步和通信机制：消息队列指针、信号量。

✓ 外存地址：由于内存紧张，将进程挂起时所在的外存地址。

✓ 家族信息

✓ 链接指针：本进程所在队列中的下一个进程的PCB的首地址。

2.2.3 进程控制块的组织结构

1. PCB队列

为了方便查找，通常将处于不同状态下的各进程PCB分别组成队列，队列的队首指针放入一个系统表中。

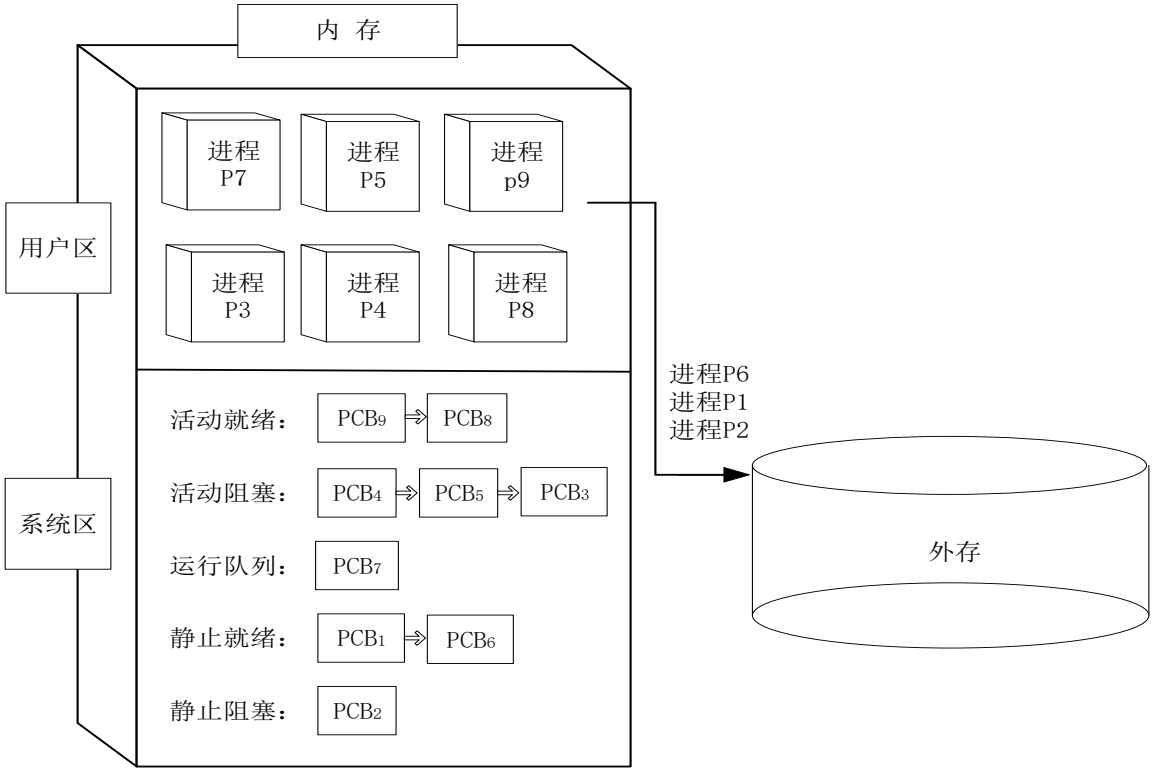


图2-10 PCB队列结构

2.PCB表

系统先将所有的PCB组织在一张PCB表中，然后再根据各进程的状态建立相应的索引表，在每个索引表的表目中，记录具有相应状态的某个PCB在PCB表中的地址。而把各索引表在内存的首地址记录在内存的一些专用单元中。

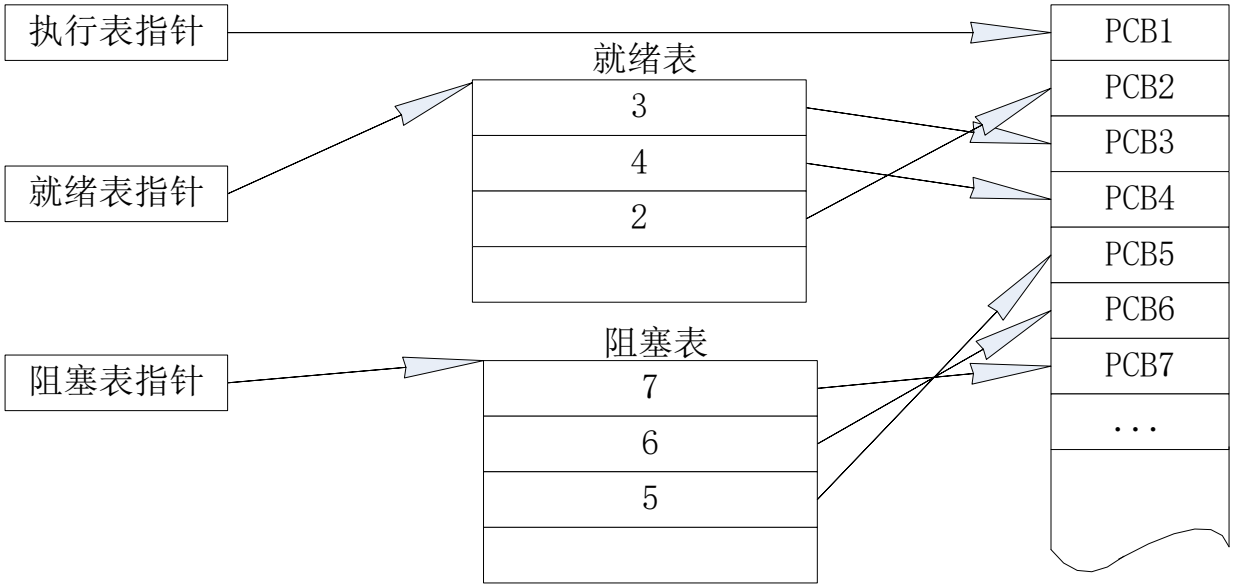


图2-11 PCB表示意图

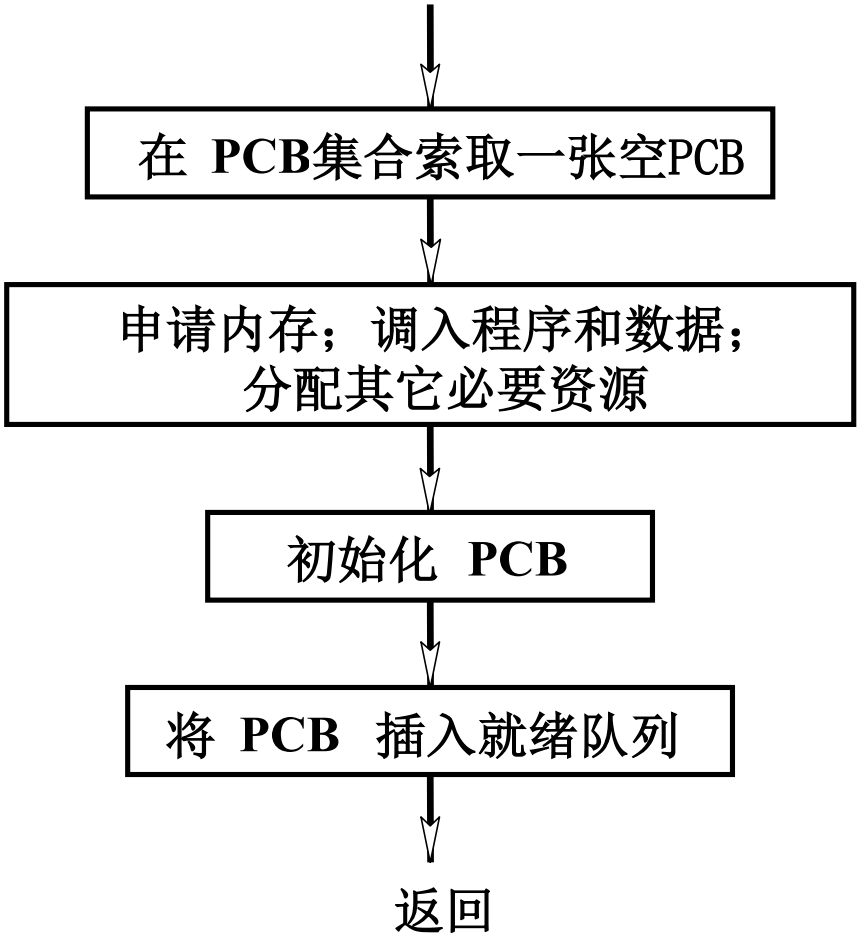
2.3 进程控制

2.3.1 进程的创建

1. 引起创建进程的事件：

- 用户登录：当终端用户登录时，由系统创建用户进程；
- 作业调度：批处理系统中，作业调度程序为选出的作业创建进程；
- 提供服务：系统为合法用户建立服务进程；
- 应用请求：进程运行时可以创建子进程来协同完成任务。

2.进程的创建过程：

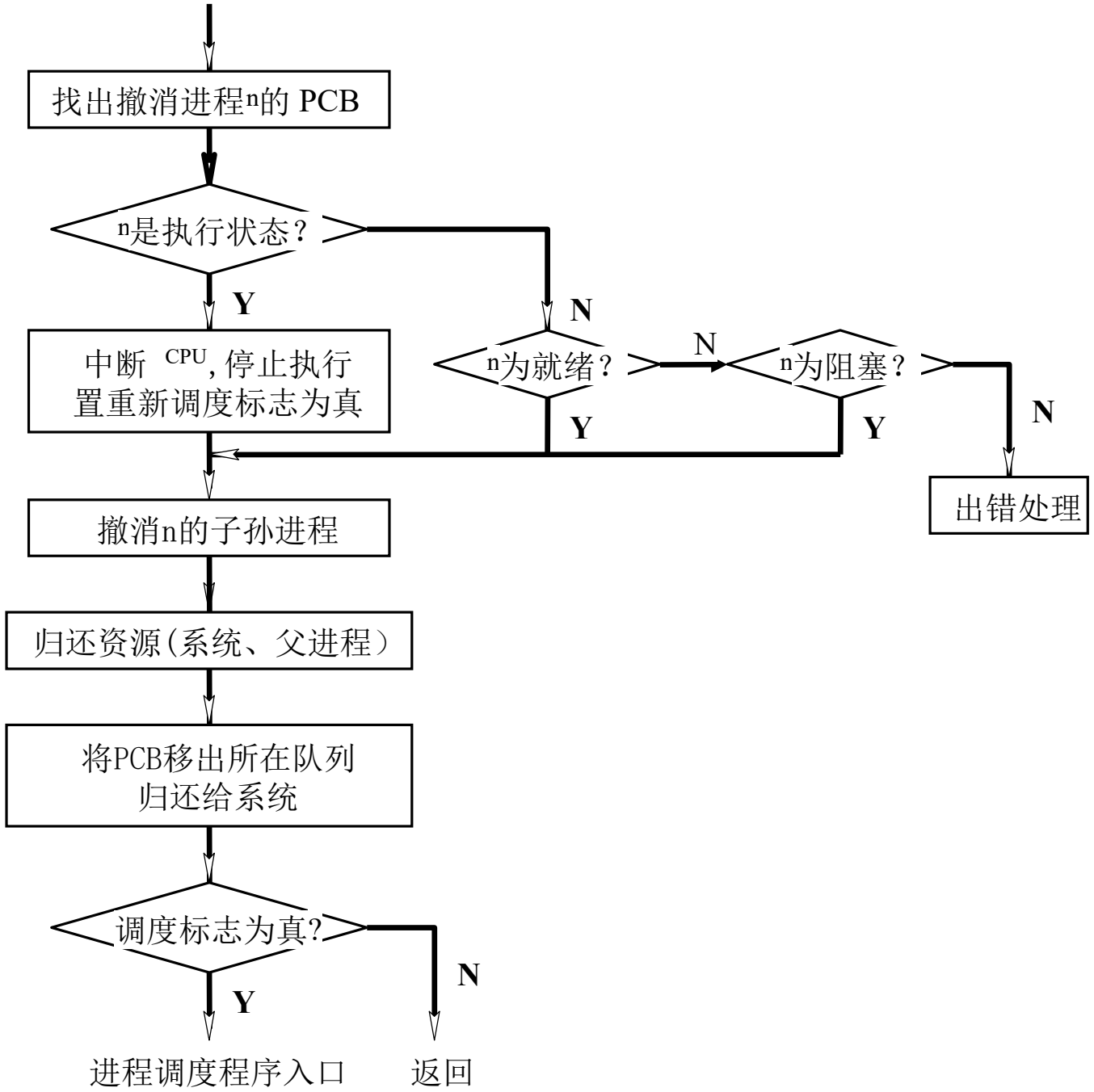

在 PCB集合索取一张空PCB
申请内存；调入程序和数据；
分配其它必要资源
初始化 PCB
将 PCB 插入就绪队列
返回

2.3.2 进程的终止

1. 引起进程终止的事件：

- 正常结束：进程完成任务，正常结束时被撤消；
- 异常结束：进程运行出现故障及错误时，被迫终止运行而被撤消；
- 外界干预：进程运行时因外界干预而撤消，如系统发生死锁时需要撤消一些进程、父进程撤消子进程等。

2.进程的终止过程：

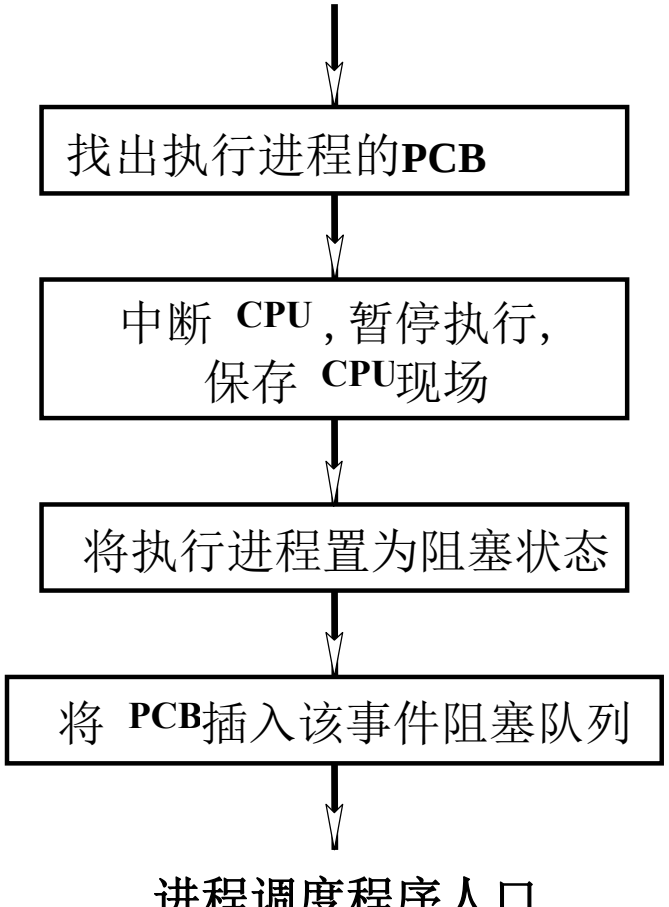

找出撤消进程n的 PCB
n是执行状态？
Y
N
中断 CPU，停止执行
置重新调度标志为真
n为就绪？
N
n为阻塞？
Y
Y
N
出错处理
撤消n的子孙进程
归还资源（系统、父进程）
将PCB移出所在队列
归还给系统
调度标志为真?
Y
N
进程调度程序入口
返回

2.3.3 进程的阻塞与唤醒

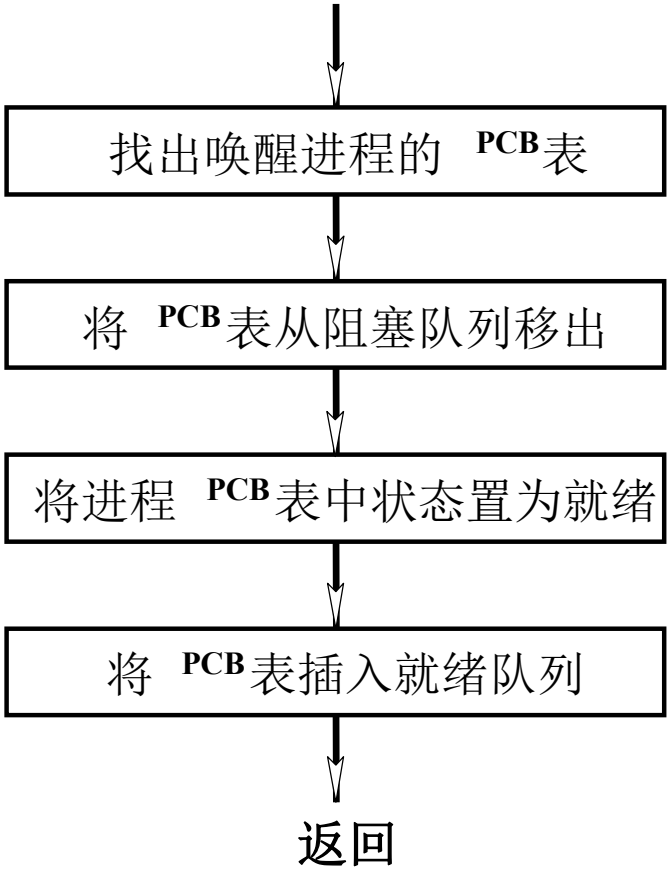
1.引起进程阻塞和唤醒的事件：

- 向系统请求共享资源失败：进程请求某共享资源，若系统已无足够的资源，通常要使其阻塞；
- 等待某种操作的完成：进程启动某操作后，必须在该操作完成后才能继续执行，应使该进程阻塞；
- 新数据尚未到达：一进程需要先获得另一进程提供的数据才能运行；
- 等待新任务的到达：某些系统进程工作时占用CPU，当无事可做时，则调用阻塞原语将自己阻塞。

2.进程的阻塞过程：


找出执行进程的PCB
中断 CPU，暂停执行，
保存 CPU现场
将执行进程置为阻塞状态
将 PCB插入该事件阻塞队列
进程调度程序人口

3.进程的唤醒过程：


找出唤醒进程的 PCB表
将 PCB表从阻塞队列移出
将进程 PCB表中状态置为就绪
将 PCB表插入就绪队列
返回

2.3.4 进程的挂起与激活

1.进程的挂起：

根据进程所处状态，挂起原语可以有三种处理：

- 完成进程从活动就绪状态到静止就绪状态的转变;
- 完成进程从活动阻塞状态到静止阻塞状态的转变;
- 若进程是执行状态，则转变为静止就绪状态。

2.3.4 进程的挂起与激活

2.挂起对象与挂起方式：

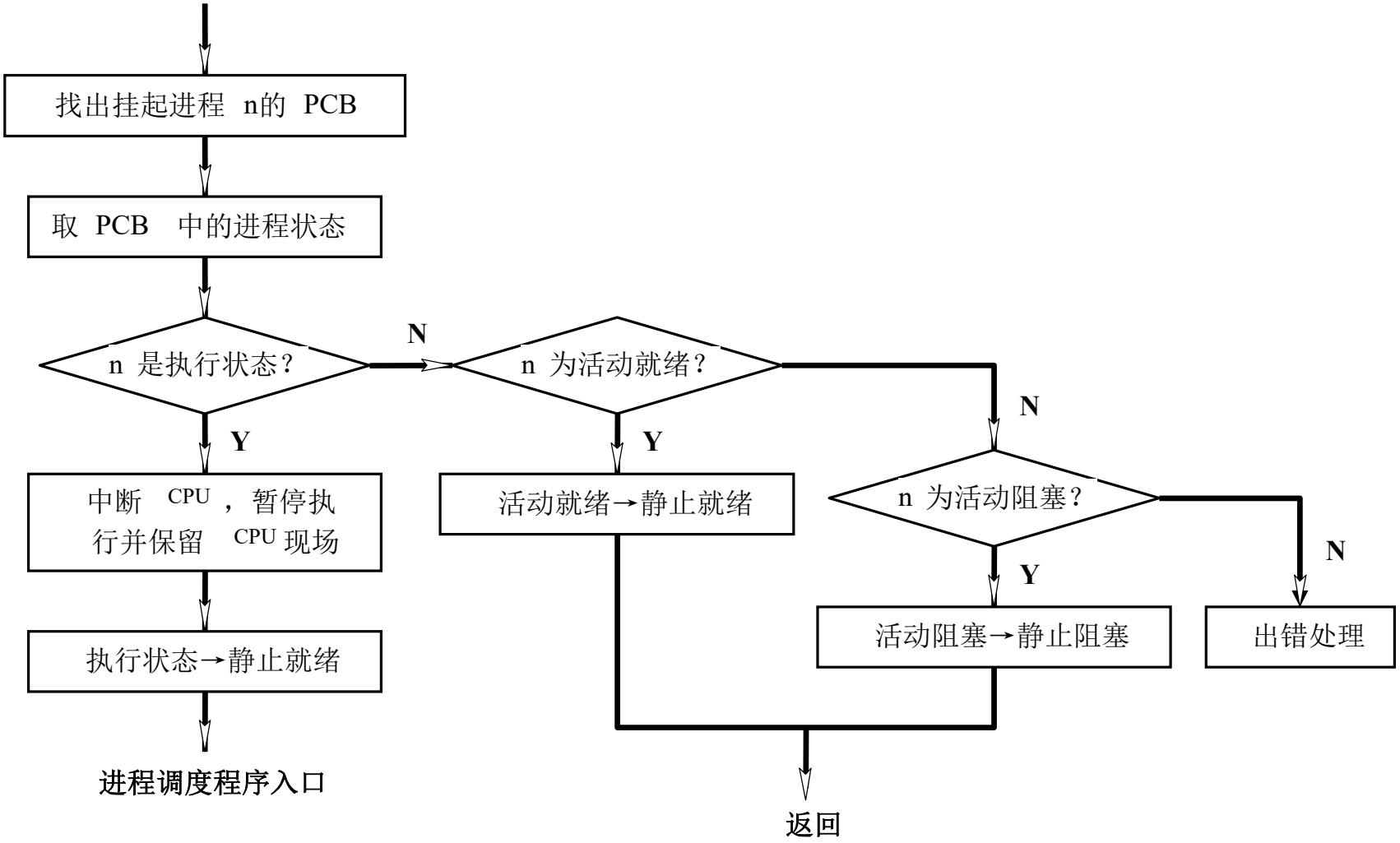
挂起对象：

- 进程请求挂起自身；
- 父进程挂起子进程。

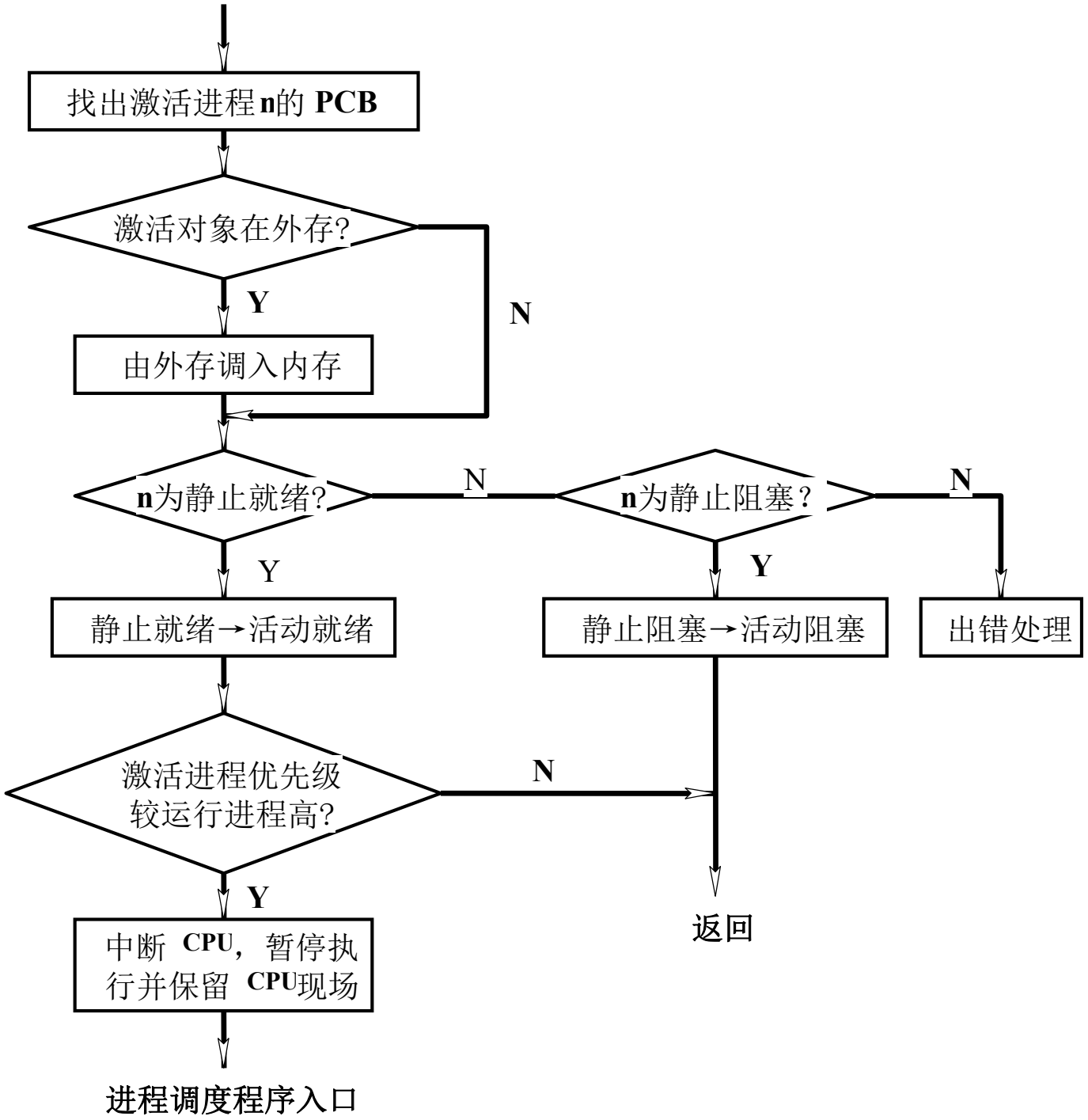
挂起方式如下：

- 挂起一个具有指定标识符的进程；
- 挂起某个进程及其所有子孙进程：采用这种挂起方式可以避免进程被挂起而其子孙进程仍在活动而带来的问题。

3.挂起过程：


找出挂起进程 n的 PCB
取 PCB 中的进程状态
n 是执行状态？
N
n 为活动就绪？
N
Y
Y
中断 CPU，暂停执行并保留 CPU 现场
活动就绪→静止就绪
n 为活动阻塞？
N
Y
执行状态→静止就绪
活动阻塞→静止阻塞
出错处理
进程调度程序入口
返回

4.激活过程：


找出激活进程n的 PCB
激活对象在外存?
Y
N
由外存调入内存
n为静止就绪?
N
n为静止阻塞？
N
Y
Y
静止就绪→活动就绪
静止阻塞→活动阻塞
出错处理
激活进程优先级
较运行进程高?
N
Y
返回
中断 CPU，暂停执
行并保留 CPU现场
进程调度程序入口

2.4 线程

2.4.1 线程的引入及基本概念

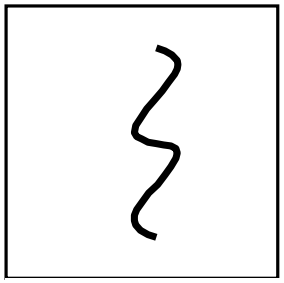
1.线程的引入

- 进程：资源分配单位（存储器、文件）和CPU调度（分派）单位。又称为"任务(task)"
- 线程：作为CPU调度单位，而进程只作为其他资源分配单位。
  - 只拥有必不可少的资源，如：线程状态、程序计数器、寄存器上下文和栈
  - 同样具有就绪、阻塞和执行三种基本状态

2. 线程的基本概念

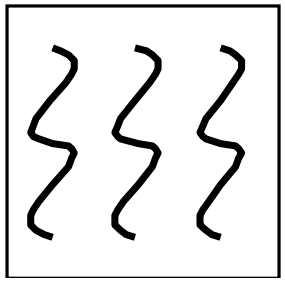
线程是进程内的一个相对独立的、可独立调度和指派的执行单元。有些操作系统中，将线程叫轻型进程（Light-weight Process）；而把传统的进程叫重型进程（Heavy-weight Process）。

- 线程的优点：减小并发执行的时间和空间开销（线程的创建、退出和调度），因此容许在系统中建立更多的线程来提高并发程度。
  - 线程的创建时间比进程短；
  - 线程的终止时间比进程短；
  - 同进程内的线程切换时间比进程短；
  - 由于同进程内线程间共享内存和文件资源，可直接进行不通过内核的通信；



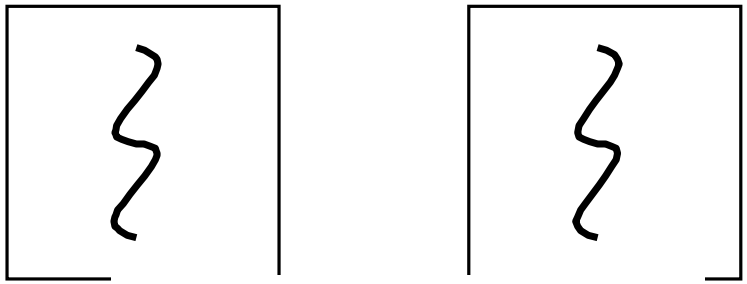
one process
one thread

MS-DOS
单用户进程、单线程



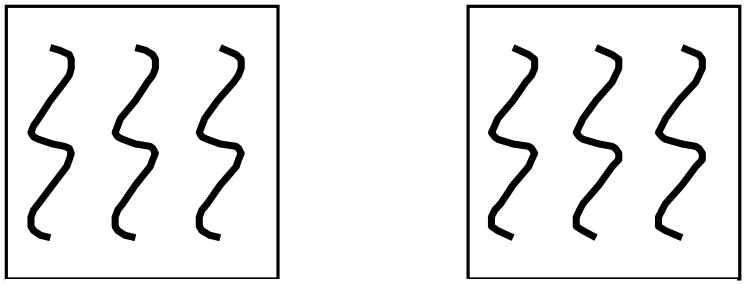
one process
multiple threads

Java运行环境
单进程、多线程



multiple processes
one thread per process

UNIX
多用户进程、每个进程一个线程



multiple processes
multiple threads per process

Windows
多进程、多线程

进程与线程的关系

2.4.2 线程的管理

1.线程控制块

- 线程状态：
  - 同样具有就绪、阻塞和运行三种基本状态
- 寄存器值：用于保存线程寄存器的上下文。
- 堆栈指针：用于保存线程的栈指针。

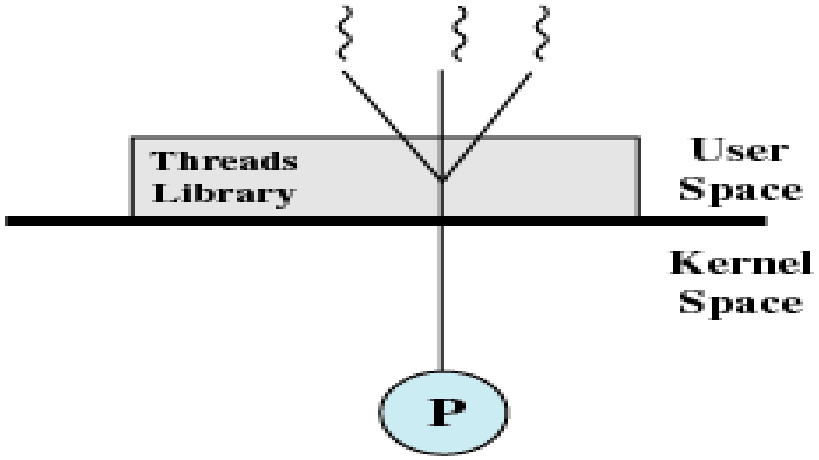
2.线程的创建和终止

3.多线程的实现

①用户级线程

用户级线程运行在用户空间，内核无需也无法感知它，调度仍以进程为单位。每个用户级线程仅需一个栈和程序计数器PC，切换速度快。当一个用户级线程被阻塞时，它会在停止之前选择并启动它的后继线程。

- 应用程序通过使用*线程库*设计成多线程程序。
  - 线程库：创建、撤销线程；在线程之间传递消息和数据；调度、切换线程
- 线程的调度也是由用户自行决定的。
- 内核不知道线程的存在，继续以进程为单位进行调度。

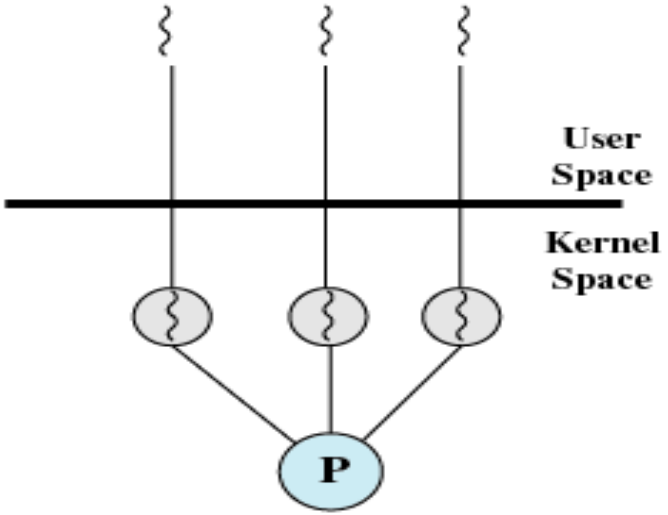


- 通常，执行应用程序时，内核为其创建一个进程。当执行该进程时，从单线程起始。若应用程序被设计成多线程的，则通过调用线程库的派生例程在该进程中创建新线程。

- 优点：
  - 所有线程管理的数据结构都在一个进程的用户地址空间，所以线程的切换无需陷入内核，故切换开销小，切换速度快。
  - 调度算法可以是进程专用的。不同的进程可以选择不同的调度算法对自己的线程进行管理，不会扰乱底层的操作系统调度器。
  - 用户级线程的实现与OS平台无关，可以在任何操作系统中运行。不需要对内核进行修改以支持用户级线程。
- 缺点：
  - 系统调用的阻塞问题。用户级线程因执行一个系统调用而阻塞时会导致整个进程中所有线程都阻塞。
  - 在多处理机系统中，同一进程中的多个线程无法调度到多个处理器上执行。

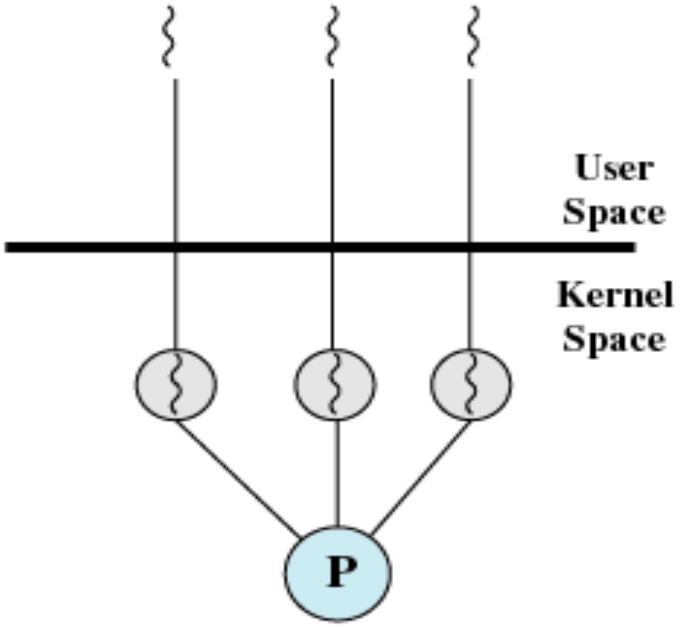
②内核级线程

一个内核支持线程可以独立工作，不需与一个用户进程联系起来。内核支持线程的创建与撤消是由系统的内部需求决定的，它共享系统的程序与全局数据，具有自己的堆栈。内核支持线程能够被单独调度并使用标准的系统同步机制。内核支持线程的创建和撤销由内核实现，运行在系统空间。



(b) Pure kernel-level

- 优点：
  - 在多处理器系统中，内核能够同时调度同一进程中的多个线程并行执行。
  - 如果进程中的一个线程被阻塞了，内核可以调度该进程中的其它线程占有处理器运行，也可以运行其它进程中的线程。
  - 内核支持线程具有很小的数据结构和堆栈，线程的切换比较快，切换开销小。
  - 内核本身也可以采用多线程技术，可以提高系统的执行速度和效率。



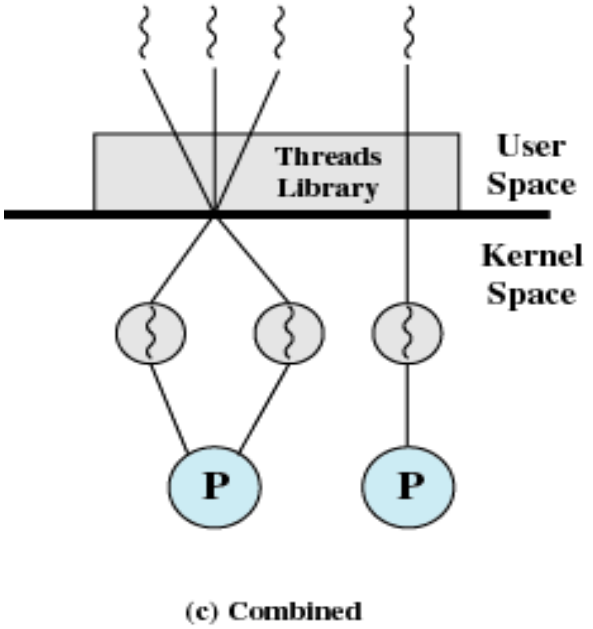
(b) Pure kernel-level

- 缺点：
  - 对于用户的线程切换而言，其模式切换的开销较大，在同一个进程中，从一个线程切换到另一个线程时，需要从用户态转到核心态进行。

③混合级线程

一个应用程序中的多个用户级线程被映射到一些内核级线程上。

小于或等于用户级线程的数目



(c) Combined

本章小结

- 前趋图，顺序执行与并发执行，
- 进程概念、基本特征、基本状态，挂起状态；
- PCB构成、组织；
- 进程创建的时机、过程；
- 进程终止的原因、过程；
- 进程阻塞与唤醒、挂起与激活的时机、过程；
- 线程，线程的优点，线程与进程的区别