操作系统
第四讲

张涛

Review

用户与操作系统的接口

作业管理

系统调用

图形用户接口

第三章

进程管理

进程管理

- 进程的概念 Process Concept
- 线程 Threads
- 处理器调度 CPU Scheduling
- 进程间同步 Process Synchronization
- 死锁 Deadlocks

进程的概念

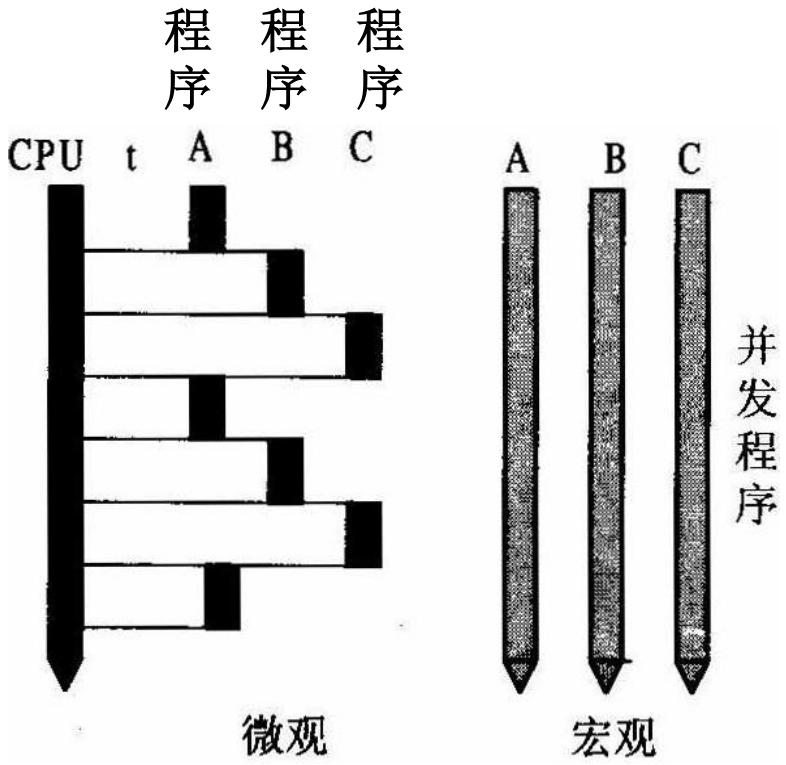
进程概念的引入

进程的表示和状态转换

进程的控制

4.1 进程概念的引入

- 顺序程序
  - 顺序性
  - 封闭性
  - 可再现性
- 并发程序
  - 间断(异步)性
  - 失去封闭性
  - 失去可再现性



并发执行的条件：达到封闭性和可再现性

与并发有关的错误

一飞机订票系统，两个终端，运行T1、T2进程

T1 :	T2:
...	...
read(x);	read(x);
if x>=1 then	if x>=1 then
x:=x-1;	x:=x-1;
write(x);	write(x);
...	...

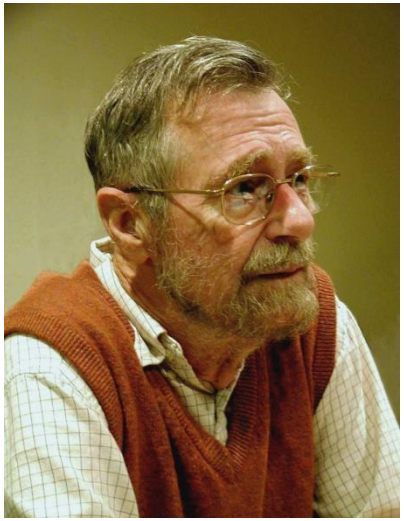
多道程序系统

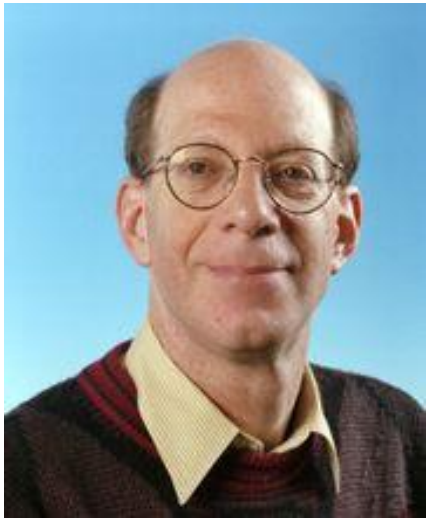
- 多道程序系统：允许多个程序同时进入内存并运行，引入目的是为了提高系统效率
  - 并行性，主存中存放多道作业并同时处于运行；
  - 制约性：各程序因资源竞争或并行程序间需要相互协同而引起的相互关系；
  - 动态性：各程序在系统中所处的状态不断变化。
- 程序概念不确切
  - 程序本身完全是一个静态的概念
  - 程序概念已不能反映系统中的并行特性

进程的概念

a program in execution

- **进程**
  - 程序在处理机上执行时所发生的活动（Dijkstra）
  - 是一个容器，该容器用以聚集相关资源（A. S. Tanenbaum）
  - **是具有独立功能的程序关于某个数据集合上的一次运行活动，是系统进行资源分配和调度的独立单位**





进程的特征

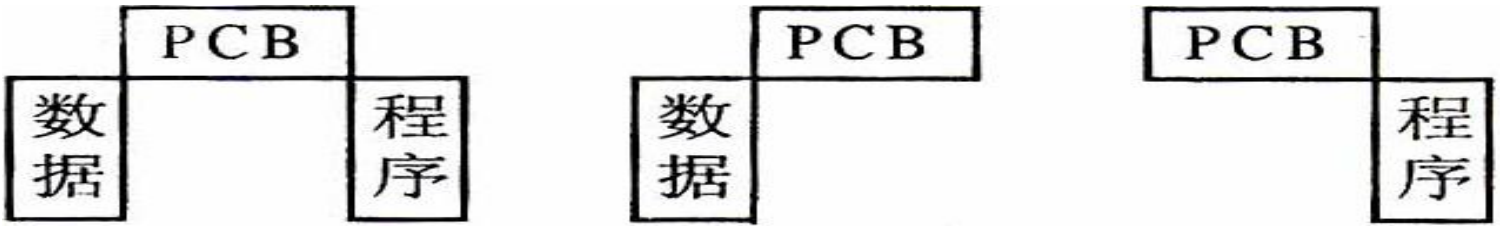
- **动态性**：进程是程序的一次执行，有着“创建”、“活动”、“暂停”、“撤消”等过程，具有一定的生命期，是动态地产生、变化和消亡的。
- **并发性**：进程之间的动作在时间上可以重叠，即系统中有若干进程都已经“开始”但又没有“结果”，称这些进程为并发进程。
- **独立性**：进程是系统调度和资源分配的独立单位，它具有相对独立的功能，拥有自己独立的进程控制块PCB。
- **异步性**：各个并发进程按照各自独立的、不可预知的速度向前推进。
- **交互性**：并发进程之间具有直接或间接的关系，在运行过程中需要进行必要的交互（同步、互斥和数据通信等），以完成特定的任务。

程序与进程之间的区别

- 程序是静态的，进程是动态的
- 进程与程序的组成不同，进程＝程序＋数据＋PCB
- 进程的存在是暂时的，程序的存在是永久的
- 一个程序可以对应多个进程，一个进程可以包含多个程序

4.2 进程的表示和状态转换

- 进程控制块PCB（Process Control Block）
  - 系统为了管理进程设置的一个专门的数据结构，用来记录进程的外部特征，描述进程的变化过程
  - 进程的组成： program+data+PCB
  - PCB是系统感知进程存在的唯一标志，进程与PCB是一一对应的

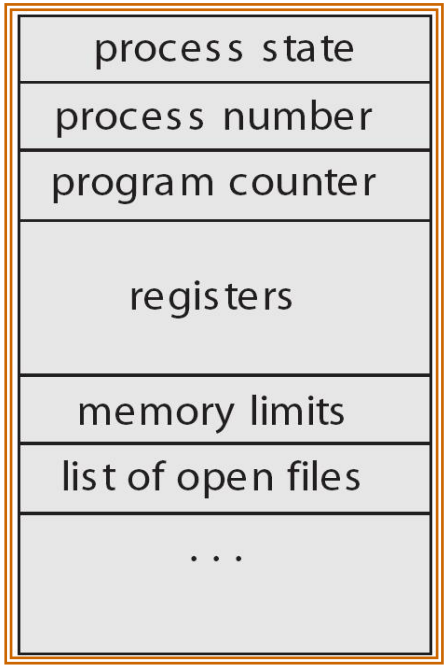


(a) (b) (c)

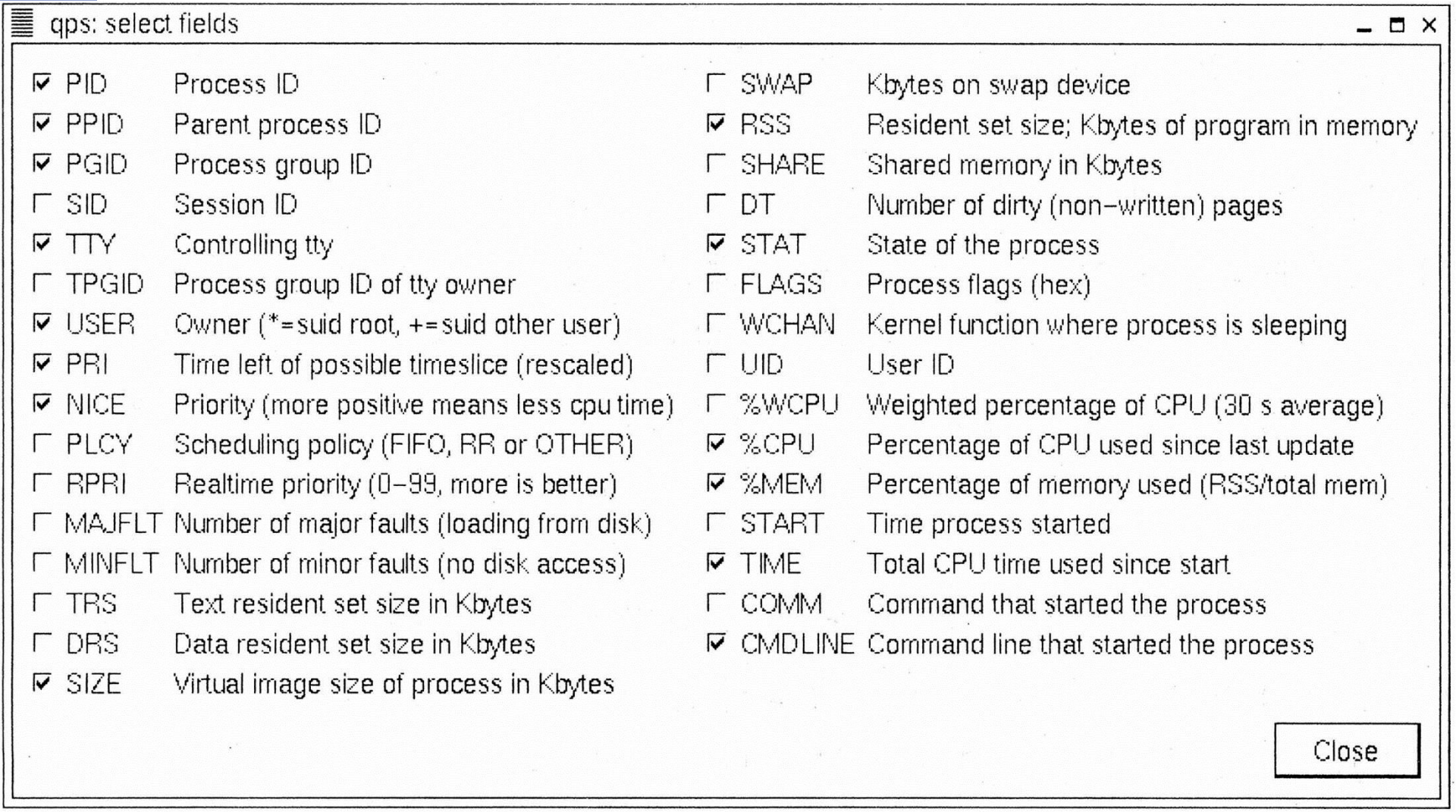
(a) 进程控制块； (b) 数据段； (c) 程序段

PCB的内容

- 进程描述信息：
  - 进程标识符(process ID)，唯一，通常是一个整数
  - 进程名，通常基于可执行文件名（不唯一）
  - 用户标识符(user ID)；进程组关系
- 进程控制信息：
  - 当前状态
  - 优先级(priority)
  - 代码执行入口地址
  - 程序的外存地址
  - 运行统计信息（执行时间、页面调度）
  - 进程间同步和通信；阻塞原因
- 所拥有的资源和使用情况：
  - 虚拟地址空间的现状
  - 打开文件列表
- CPU现场保护结构：寄存器值



Linux中的进程描述参数

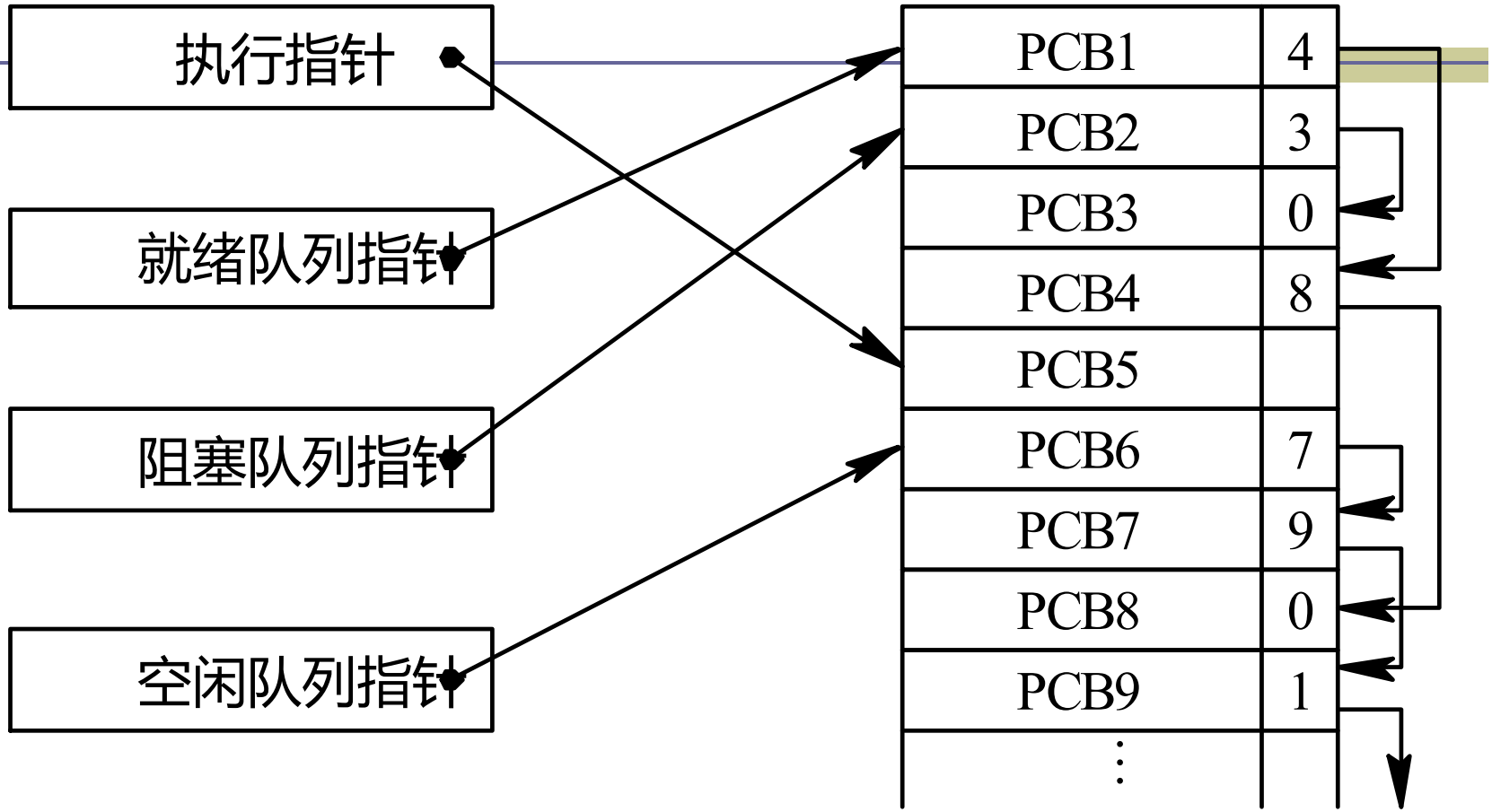


PCB是进程存在的唯一标志？

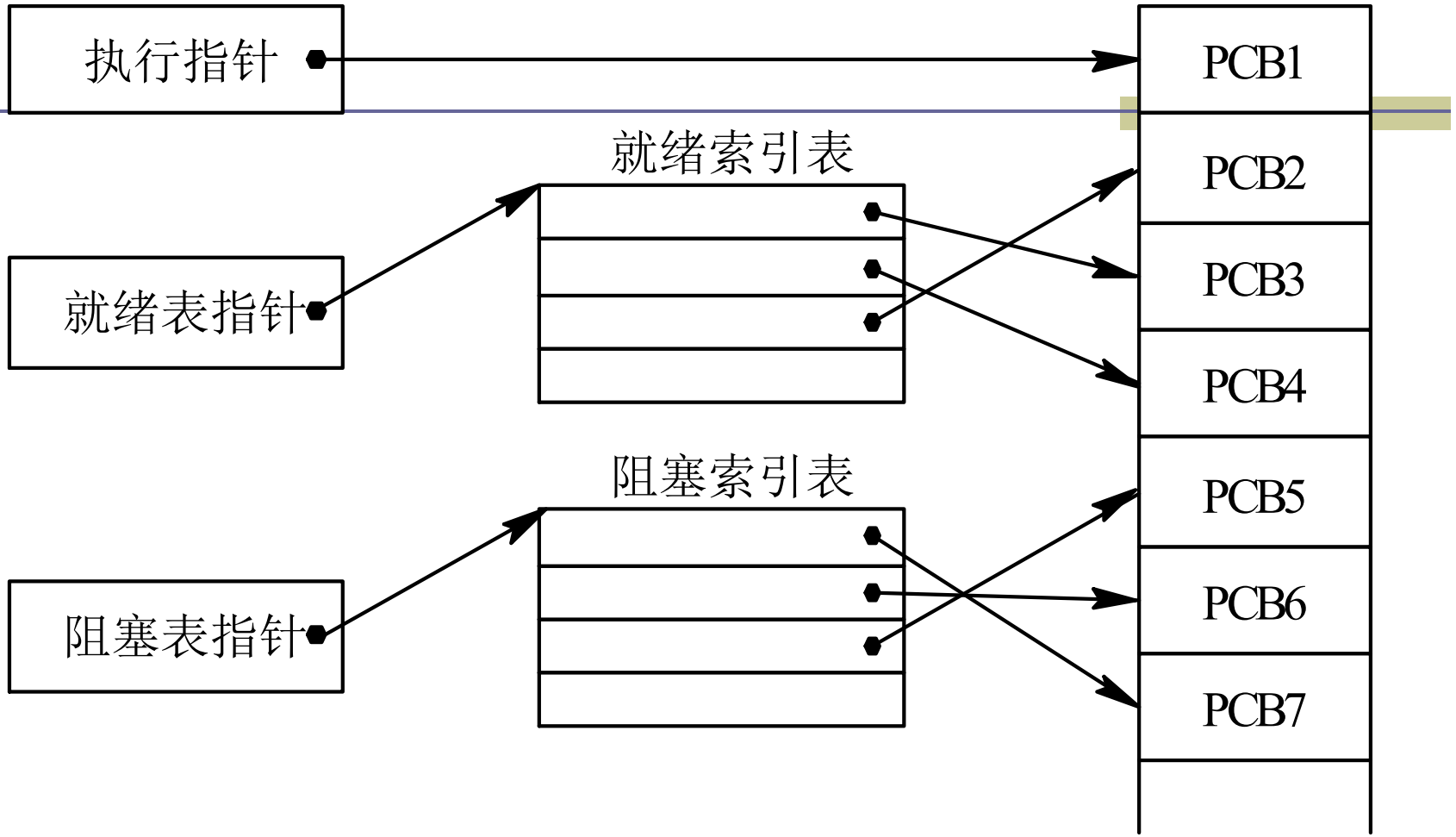
- 1）包含了进程的描述信息和控制信息，
- 2）是进程的动态特征的集中反映，
- 3）系统根据PCB而感知某一进程的存在

PCB表组织方式

- PCB表：
  - 系统把所有PCB组织在一起，并把它们放在内存的固定区域，就构成了PCB表
  - PCB表的大小决定了系统中最多可同时存在的进程个数，称为系统的并发度
- 组织方式：
  - 链表：同一状态的进程其PCB成一链表，多个状态对应多个不同的链表：就绪链表、阻塞链表
  - 索引表：同一状态的进程归入一个index表（由index指向PCB），多个状态对应多个不同的index表：就绪索引表、阻塞索引表

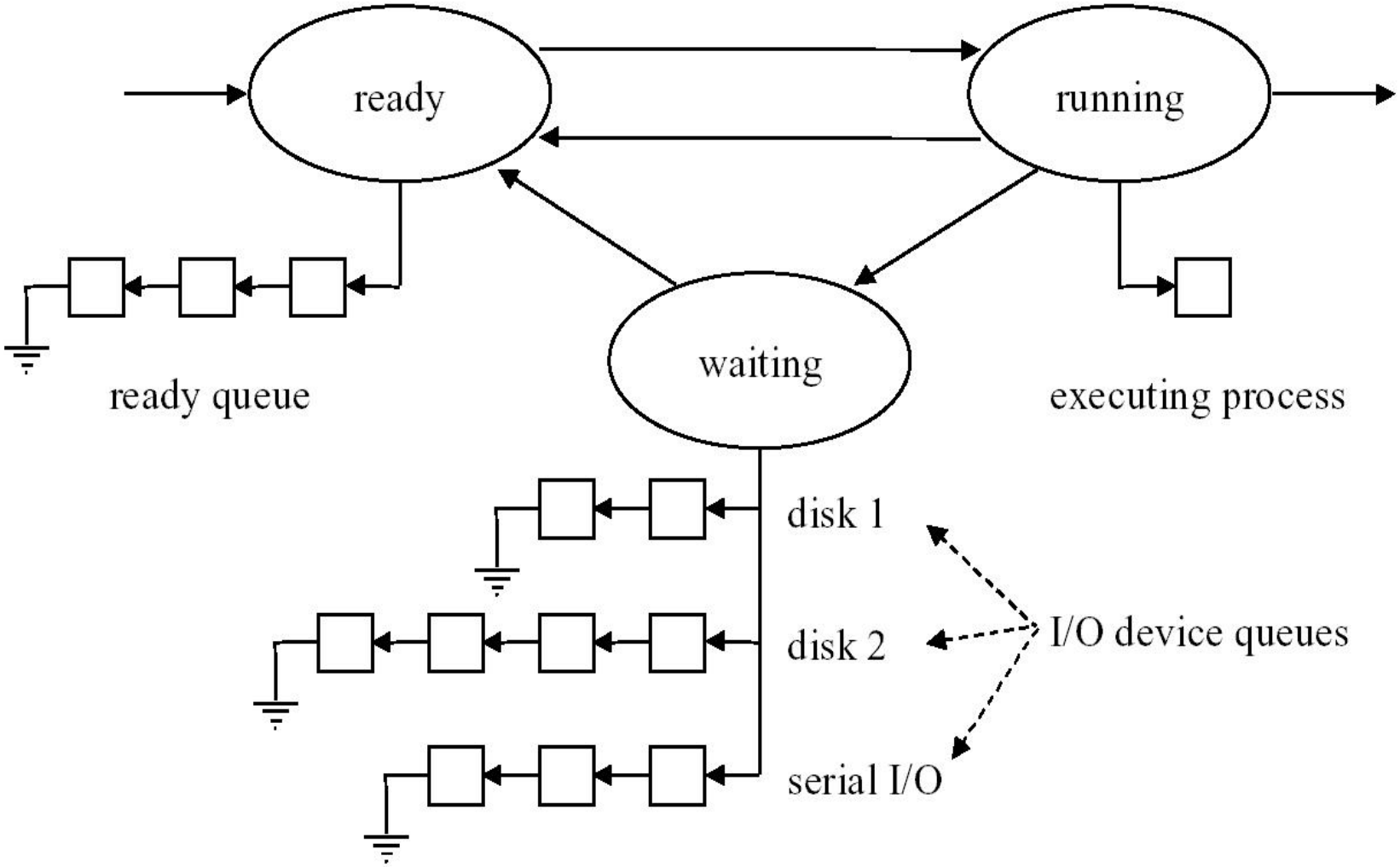

执行指针
就绪队列指针
阻塞队列指针
空闲队列指针
PCB1
PCB2
PCB3
PCB4
PCB5
PCB6
PCB7
PCB8
PCB9
4
3
0
8
7
9
0
1

PCB链表队列


执行指针
就绪索引表
就绪表指针
阻塞索引表
阻塞表指针
PCB1
PCB2
PCB3
PCB4
PCB5
PCB6
PCB7

按索引方式组织PCB

Linked List

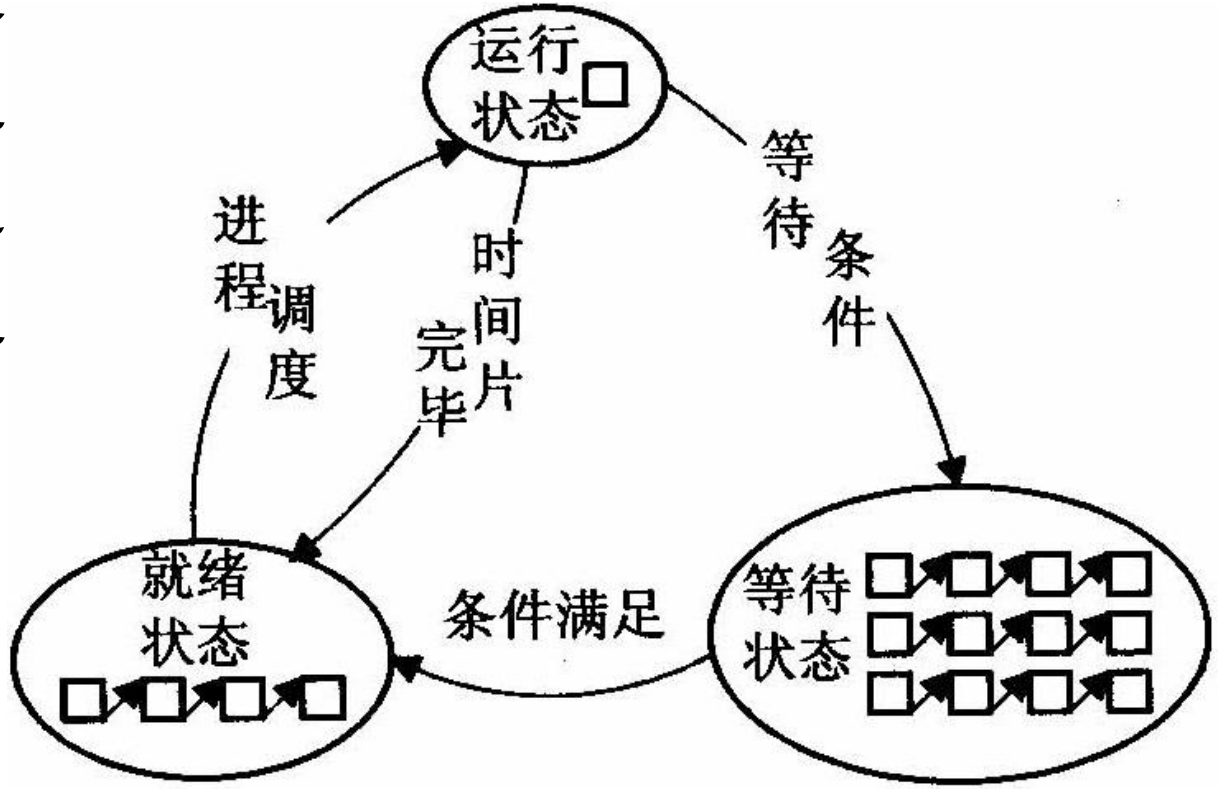

ready
running
waiting
ready queue
executing process
disk 1
disk 2
serial I/O
I/O device queues

进程的状态

- 最基本的进程状态有三种：
- 运行状态（Running），进程占有CPU，并在CPU上运行；
- 就绪状态（Ready），一个进程已经具备运行条件，但由于无CPU暂时不能运行的状态（当调度给其CPU时，立即可以运行）；
- 等待状态（阻塞状态，Blocked），阻塞态、封锁态、睡眠态，指进程因等待某种事件的发生而暂时不能运行的状态（即使CPU空闲，该进程也不可运行）

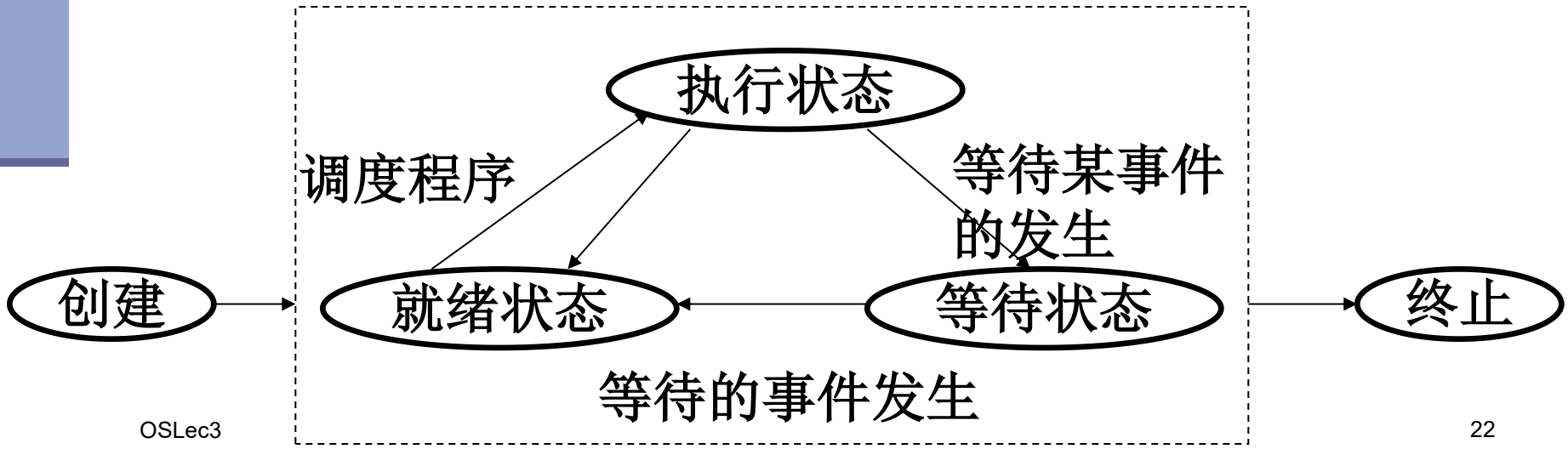
进程状态转换

- 就绪—运行
- 运行—就绪
- 运行—等待
- 等待—就绪

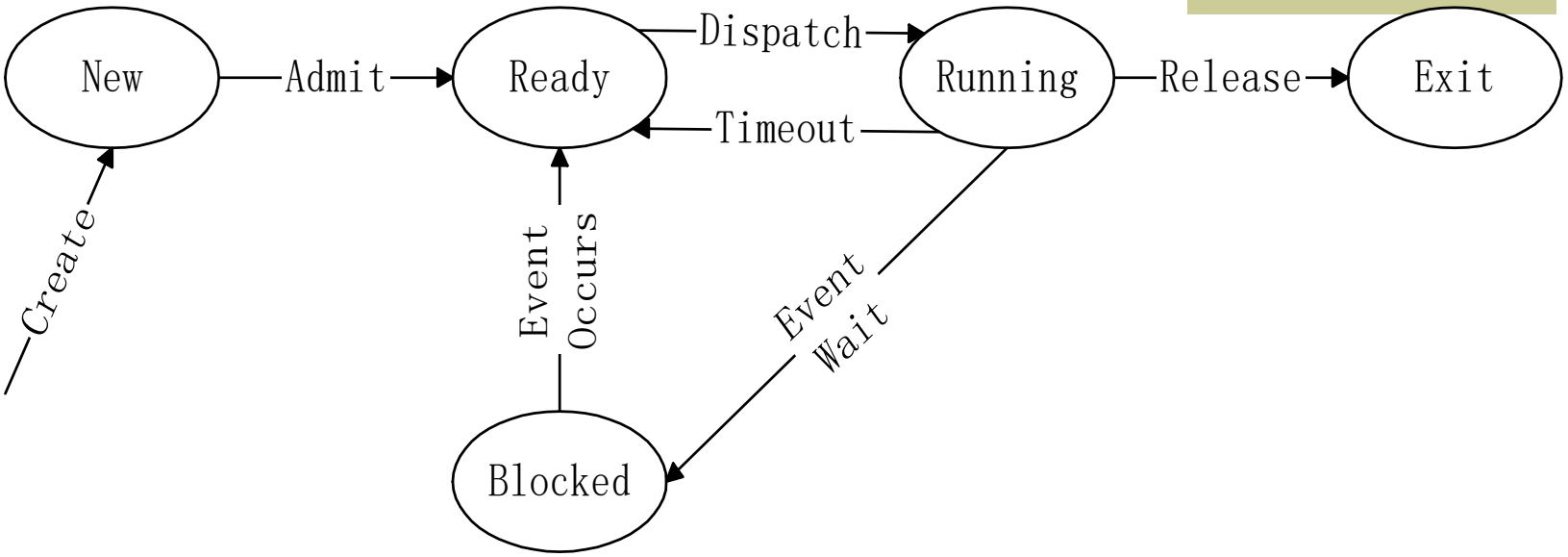


五状态进程

- 增加：
  - 创建状态(New)：进程刚创建，但还不能运行如：分配和建立PCB表项、建立资源表格并分配资源，加载程序并建立地址空间表。
  - 结束状态(Exit)：进程已结束运行，回收除PCB之外的其他资源，并让其他进程从PCB中收集有关信息。



五状态进程转换图

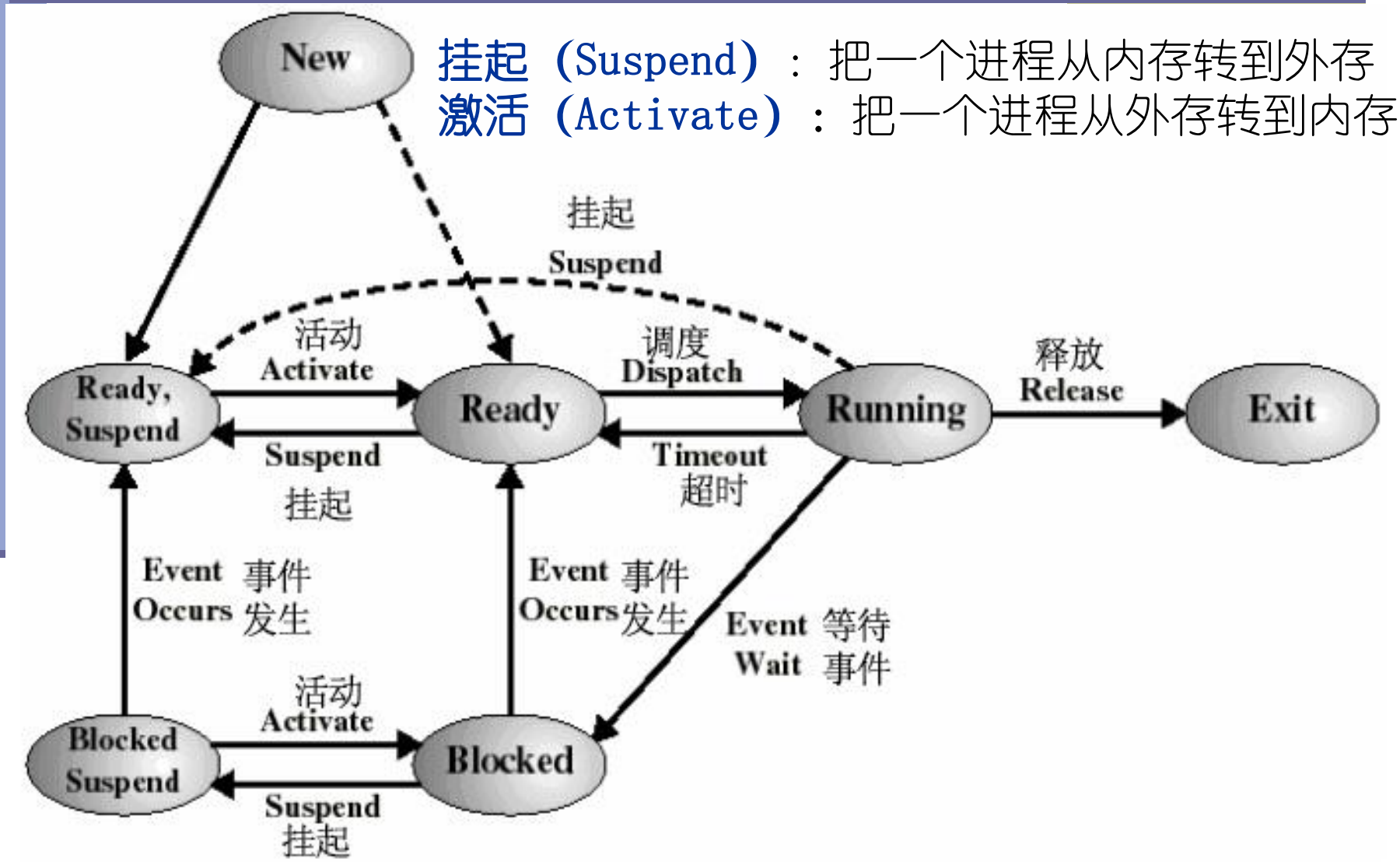


- 创建新进程Create
- 提交（收容）Admit
- 调度运行(Dispatch)
- 释放(Release)
- 超时（Timeout）
- 事件等待（Event Wait）
- 事件出现（Event Occurs）

挂起进程模型

- 挂起（Suspend）：一些低优先级进程可能等待较长时间而被对换至外存，为运行进程提供足够内存。
  - 阻塞挂起（Blocked, suspend）：进程在外存并等待某事件的出现；
  - 就绪挂起（Ready, suspend）：进程在外存，但只要进入内存，即可运行；

七状态进程转换图

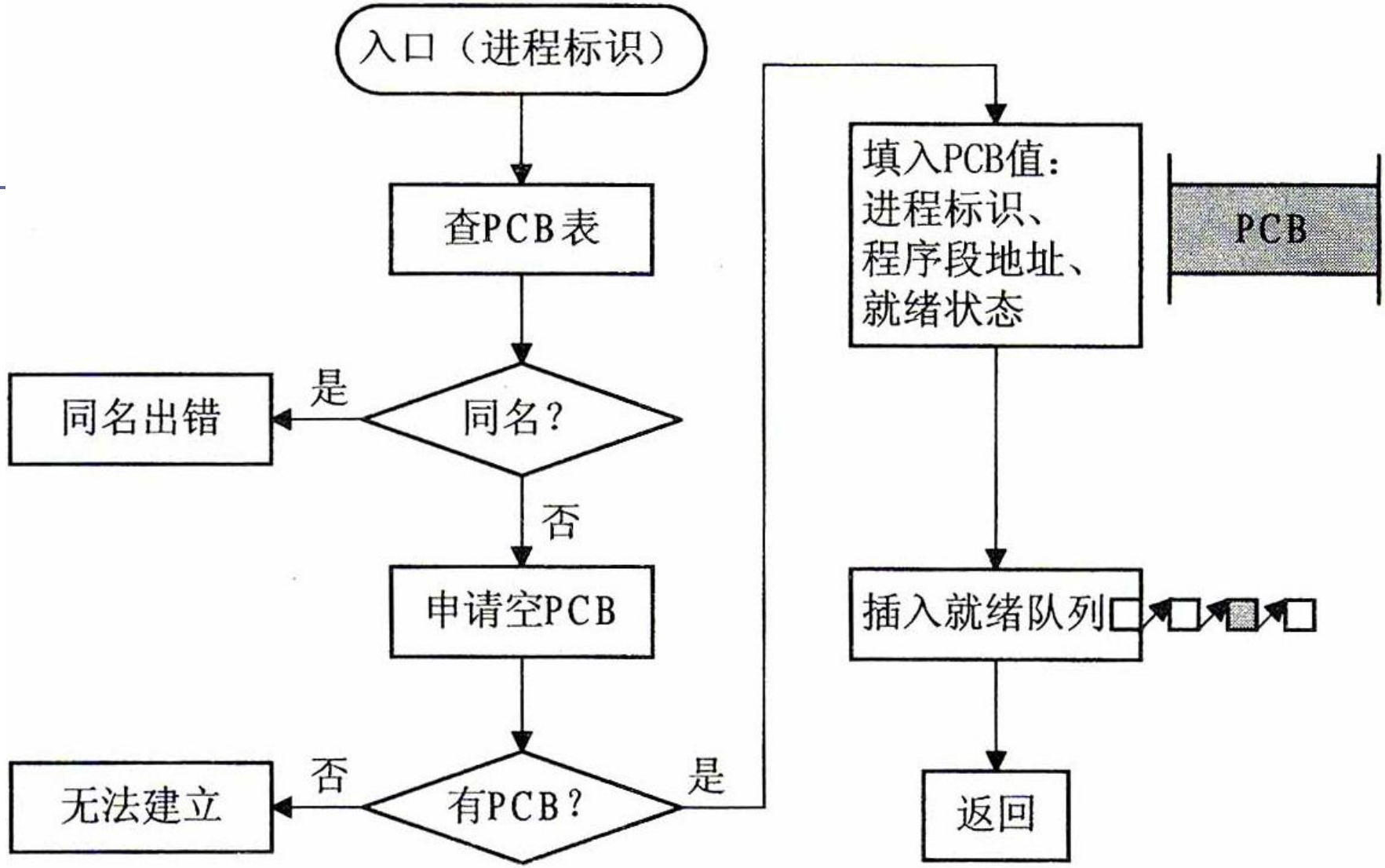

New
挂起 (Suspend)：把一个进程从内存转到外存
激活 (Activate)：把一个进程从外存转到内存
挂起
Suspend
活动
Activate
调度
Dispatch
释放
Release
Ready, Suspend
Ready
Running
Exit
Suspend
挂起
Timeout
超时
Event 事件
Occurs 发生
Event 事件
Occurs 发生
Event 等待
Wait 事件
活动
Activate
Blocked Suspend
Blocked
Suspend
挂起

4.3 进程的控制

- 创建、撤消进程以及完成进程各状态之间的转换，由具有特定功能的原语完成
- “原语”是由若干条机器指令构成、完成一种特定功能的程序段；这段程序在执行期间不允许被分割，必须一次执行完。
- 进程控制原语：
  - 进程创建原语
  - 进程撤消原语
  - 进程状态转换原语

进程的建立

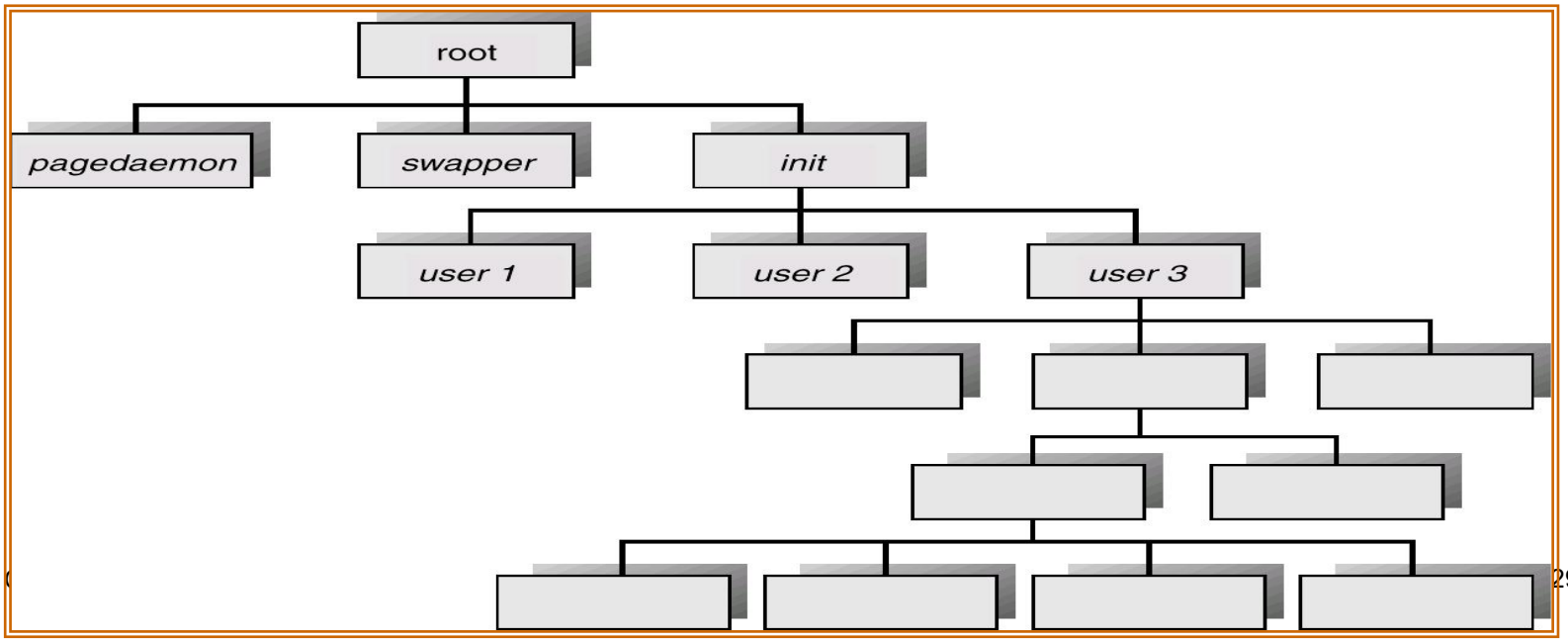
- 何时创建：
  - 用户登录、 作业调度、提供服务、应用请求
- 进程创建的基本过程
  - 首先从空闲的PCB集合中申请一个新的PCB，同时获得该进程的内部标识；
  - 然后向该PCB中填写各种参数；
  - 把该进程的状态设置成就绪状态，并将该PCB插入到就绪队列中。


入口（进程标识）
查PCB表
同名？
是
同名出错
否
申请空PCB
有PCB？
否
无法建立
是
填入PCB值：
进程标识、
程序段地址、
就绪状态
PCB
插入就绪队列
返回

进程创建原语

继承家族树

- 资源分配严格，子进程只能继承父进程所拥有的资源，便于管理；
- 系统可根据需要赋予进程不同的控制权，并可以把一个任务分解成若干个进程来完成，具有较好的灵活性；
- 树形结构层次清晰，关系明确。



Unix Example

- fork system call creates new process
- exec system call used after a fork to replace the process' memory space with a new program

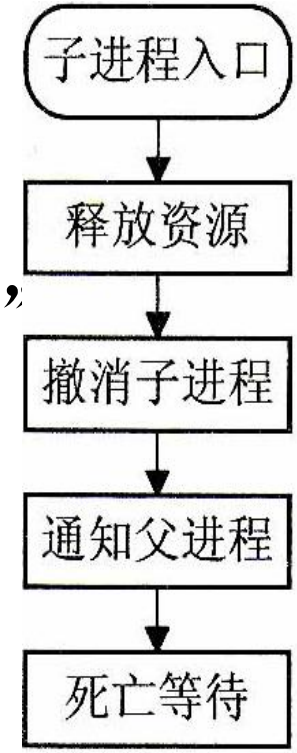
```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char *argv[])
{
int pid;
    /* fork another process */
    pid = fork();
    if (pid < 0) { /* error occurred
    */
        fprintf(stderr, "Fork
    Failed");
        exit(-1);
    }
    else if (pid == 0) { /* child
    process */
        execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
        /* parent will wait for the
    child to complete */
        wait(NULL);
        printf("Child Complete");
        exit(0);
```

进程的撤销

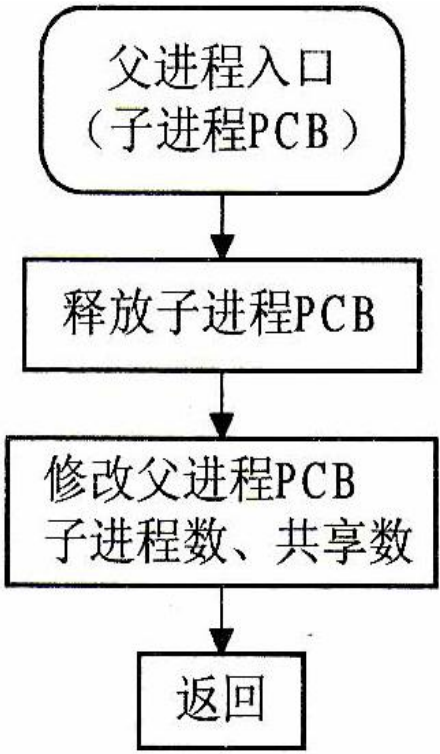
- 引起进程终止(Termination of Process)的事件
  - 正常结束
  - 异常结束
  - 外界干预
- 撤消进程的两种策略：
  - 撤消指定进程
  - 撤消该进程及其所有子孙进程

进程终止的基本过程

- 找到相应进程的PCB；
- 若进程正处于执行状态，则立即停止，设置重新调度标志；
- 撤消属于该进程的所有“子孙”进程；
- 释放被撤消进程的所有资源；
- 释放进程的PCB；
- 若调度标志为真，则进行重新调度



(a)子进程撤销



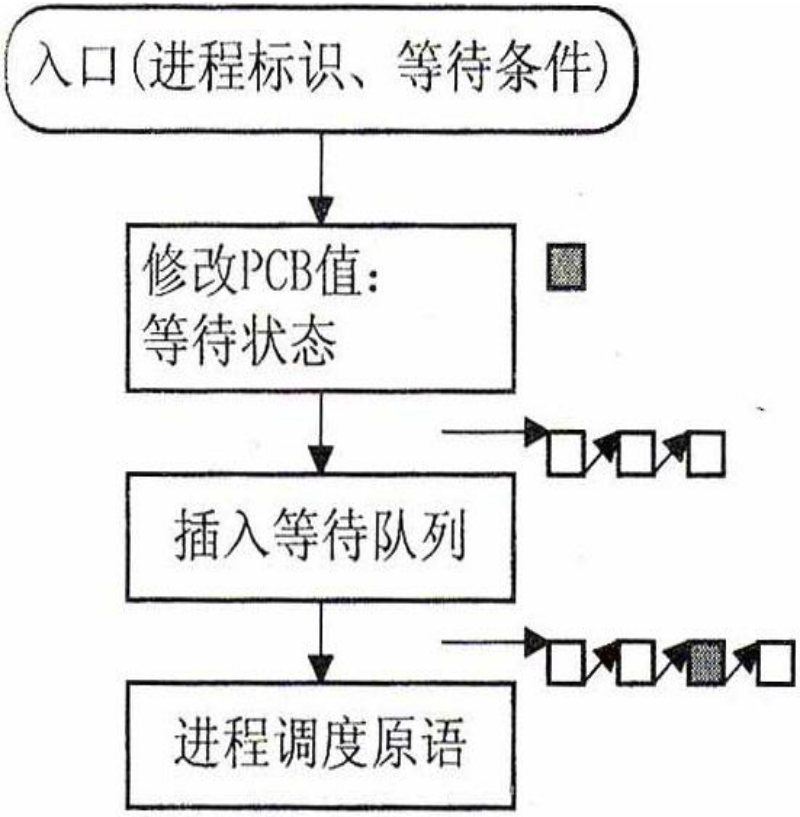
（b）父进程撤销

进程的状态转换原语——等待

- 进程的等待
  - 在进程的运行过程中，如果申请某一种条件而没有被满足，进程不得不中止当前的运行，进程等待原语就会被激活
- 进程等待原语
  - 使调用该原语的进程变为等待状态；
  - 将指定的进程变为等待状态；
  - 将某进程及其所有子孙进程变为等待状态。

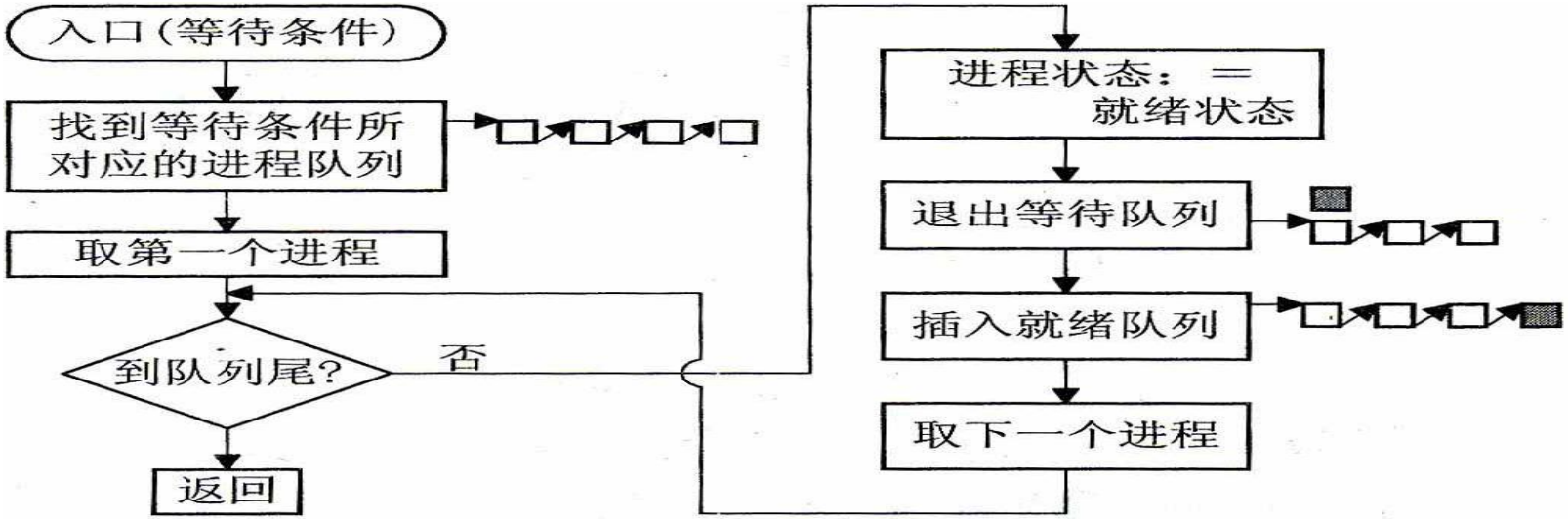
进程等待的基本过程

- 找到相应进程的PCB；
- 如果该进程为执行状态，则保护其现场，将其状态改变为等待状态，停止运行，并把该PCB插入到相应的等待队列中去；
- 若为就绪状态，则将其状态修改为等待状态，把它移出就绪队列，并插入到等待队列中去



进程的状态转换原语——唤醒

- 进程因等待某事件的发生而处于等待状态，当等待事件发生后，就要用唤醒原语将其唤醒。
- 唤醒原语的基本操作：
  - 在等待队列中找到相应进程的PCB，将其从等待队列中移出；
  - 置其状态为就绪状态，然后把该PCB插入就绪队列中；
  - 等待调度程序调度。



进程的状态转换原语.....

- 进程调度原语
  - 找到就绪队列的首指针，按照调度算法所规定的选择原则（比如优先级法）选中一个进程，将该进程的PCB中的状态由就绪状态改变为运行状态，然后使其退出就绪队列，恢复该进程的现场参数，该进程便进入运行状态。
- 进程挂起原语
  - 检查被挂起进程的状态，若处于活动就绪状态，便将其改为静止就绪；对于活动阻塞状态的进程，则将之改为静止阻塞。为了方便用户或父进程考查该进程的运行情况而把该进程的PCB复制到某指定的内存区域。

进程的状态转换原语

- 进程激活原语
  - 先将进程从外存调入内存，检查该进程的现行状态，若是静止就绪，便将之改为活动就绪；若为静止阻塞便将之改为活动阻塞。假如采用的是抢占调度策略，则每当有新进程进入就绪队列时，应检查是否要进行重新调度，即由调度程序将被激活进程与当前进程进行优先级的比较，如果被激活进程的优先级更低，就不必重新调度；否则，立即剥夺当前进程的运行，把处理机分配给刚被激活的进程

Linux中的进程控制原语

- 在Linux系统中，进程控制的原语有：
  - 进程建立fork、
  - 进程监控ps、
  - 进程优先级的确定nice、
  - 进程等待lock、
  - 进程唤醒wakeup、
  - 进程终止kill等。
- 在Linux中，系统引导时会自动建立一个进程，称为进程0，这个进程是所有进程的祖先，负责完成进程的调度。然后进程0建立自己的子进程：进程1。除进程1外，进程0将建立其他许多与系统管理有关的进程。

What you need to do?

- 复习课本3.1~3.3节的内容
- 课后作业：
- P91，习题1、2、3、6