

第2章 进程管理



- 操作系统中最核心的概念是**进程**：对正在运行程序的一个抽象。
- 操作系统的其他所有内容都是围绕着**进程**的概念展开的。
- 在一个多道程序设计系统中，CPU在各**进程**之间切换。
- 处理机管理→**进程管理**



本章主要内容

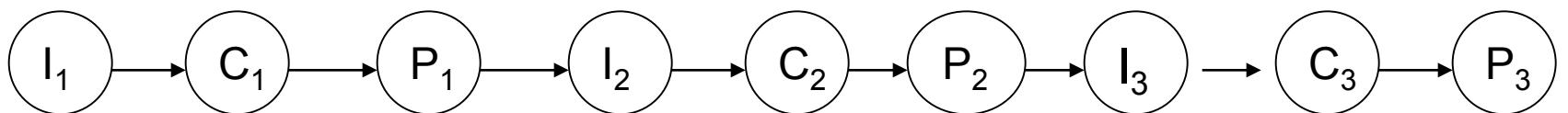
1. 进程的引入和概念
2. 进程的描述：PCB、状态、
3. 进程的控制：创建、撤销、阻塞、唤醒...
4. 处理机的调度：分配CPU给某一进程
5. 线程的引入



2.1 进程的引入及其概念

程序的顺序执行：在任何时刻，机器只执行一个操作，只有在前一个操作执行完后，才能执行后继操作。

[例]作业*i*的输入操作、计算操作和打印操作分别用 I_i 、 C_i 、 P_i 表示。则顺序执行过程为：





2.1 进程的引入及其概念

1. 程序的顺序执行：计算机每次只运行一个程序，只有在前一个运行完之后再运行下一个。如，单道批处理系统。
- 封闭性：程序在运行时独占全机资源。因此，这些资源的状态只能由这个运行的程序决定和改变。不受外界因素影响。
- 可再现性：只要初始条件相同，无论程序是连续运行，还是断断续续地运行，程序的执行结果与其执行速度无关。



- 优点：由于顺序程序的封闭性和可再现性，
为程序员调试程序带来了很大方便。
- 缺点：由于资源的独占性，使得系统资源
利用率非常低。



2. 程序的并发执行

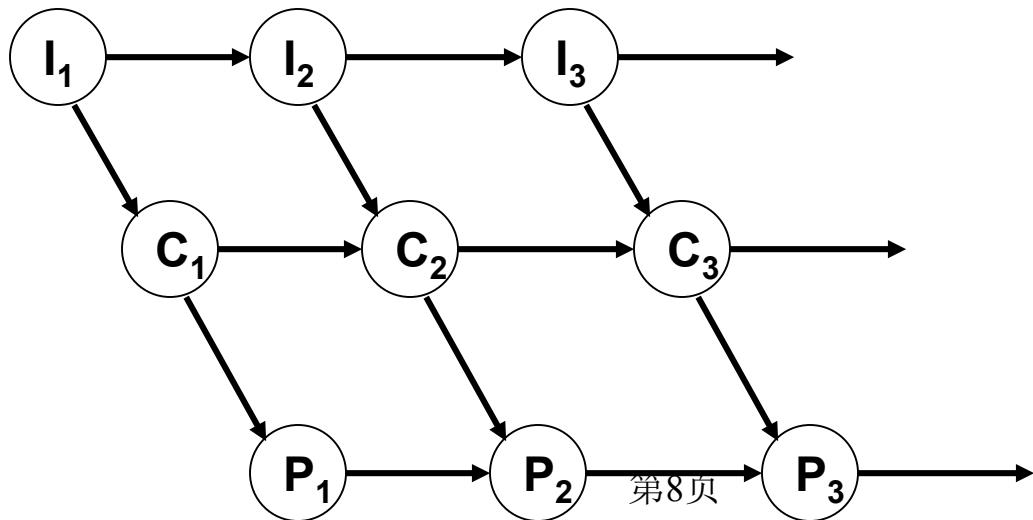
- 并发执行：计算机同时运行几个程序。
CPU要不断地在几个程序之间切换。
- 以资源共享为条件。
- 增强计算机系统的处理能力，提高资源利用率。



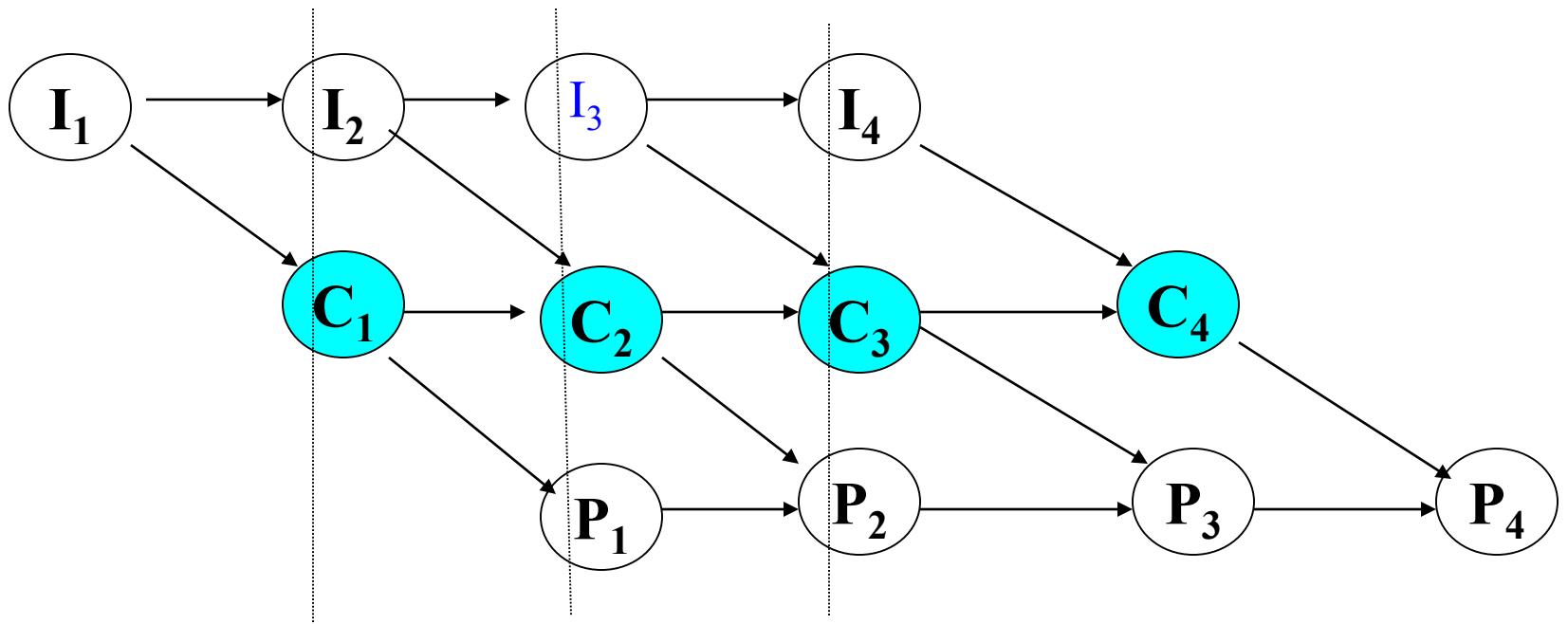
前趋图

它是一个有向无循环图，图中每个节点都可表示一条语句、一个程序段或一个进程，节点间的有向边表示两个节点之间存在的偏序或前趋关系。

其中，没有前趋的节点称为初始节点，没有后继的节点称为终止结点。如下图所示。



[例] 作业 i 的输入、计算和输出操作分别用 I_i 、 C_i 、 P_i 表示。虽然同一作业中的输入、计算和输出必须顺序执行，但对一批作业而言，情况就不同了。



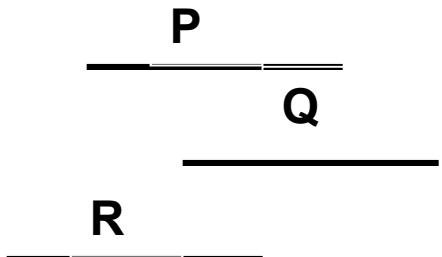
并发执行

并发执行



并发执行的表示

- 如下图所示的三个程序段就是并发执行的程序段。可用如下语句来表示：





程序并发执行的特征：(1,2,3)

(1) 失去了程序的封闭性和可再现性

在并发执行时，多个程序共享系统中的各种资源，因而这些资源的状态将由多个程序来改变，致使程序的运行失去了封闭性；由于失去了封闭性，也将导致失去其可再现性。



[例] 有两个循环程序A和B，共享一个变量N。

A每执行一次时都要做 $N=N+1$; B每执行一次时都要做 $\text{print}(N)$, $N=0$ 。并以不同的速度运行。

program A:

$N = n ;$

...

$N=N+1;$

...

...

program B:

$N = n ;$

...

$\text{print}(N);$

$N=0;$

...



假如系统中有两道程序A和B，这两个程序都正在打印机上输出。程序输出完成后将打印机释放。

假定系统用变量N来记录打印机的可用数量。假定N的当前值为1，也即还有一台空闲。



当A、B程序释放打印机后，系统的打印机数量为3台（即N=3）。但由于两个程序并发执行，各程序执行释放操作是随机的，有可能使得N的值结果不正确。

下面是两个程序释放打印机的操作：

```
int N=1;  
program A:  
begin  
    execute program A;  
    释放打印机: N=N+1 ;  
    .....  
end;
```

```
program B:  
begin  
    execute program B;  
    释放打印机: N=N+1 ;  
    .....  
end;
```



“ $N=N+1$ ”的操作分解成机器指令为：

取N到寄存器A；

$A+1 \rightarrow A$ ；

存A到N；

由于程序A和B都以各自独立的速度向前推进，故程序A执行 $N=N+1$ 的操作与程序B执行 $N=N+1$ 操作是随机的。而这两个程序执行N的序列可用表描述如下：



时间	T0	T1	T2	T3	T4	T5
程序A	$N \rightarrow A_1$	$A_1 + 1 \rightarrow A_1$	$A_1 \rightarrow N$	$N \rightarrow A_2$		
程序B					$A_2 + 1 \rightarrow A_2$	$A_2 \rightarrow N$
N的值	1	1	2		2	3

(a) 顺序执行

时间	T0	T1	T2	T3	T4	T5
程序A	$N \rightarrow A_1$			$A_1 + 1 \rightarrow A_1$		$A_1 \rightarrow N$
程序 B		$N \rightarrow A_2$	$A_2 + 1 \rightarrow A_2$		$A_2 \rightarrow N$	
N 的值	1	1	1	1	2	2

(b) 交叉执行

时间	T0	T1	T2	T3	T4	T5
程序 A		$N \rightarrow A_1$		$A_1 + 1 \rightarrow A_1$		$A_1 \rightarrow N$
程序 B	$N \rightarrow A_2$		$A_2 + 1 \rightarrow A_2$		$A_2 \rightarrow N$	
N 的值	1	1	1	1	2	2

(c) 交叉执行 第16页



(2) 并行执行的程序间产生了相互制约关系

因共享资源或协调完成同一任务而引起的。

[例] 并发执行的程序A和B共享一台打印机。
A和B之间就产生了间接制约关系。

[例] type a. c | more

这条命令就需要管道符号两边的程序协作完成任务。是一种直接制约关系。



(3) 程序与CPU执行的活动之间不再一一对应

- 程序是完成特定功能的指令序列，是静态的
- CPU执行的活动是一个动态概念，是程序的执行过程。

[例] 分时系统中，多个用户都调用C编译器对自己的源程序进行编译。实际系统只保留一个编译程序，而CPU正在为多个用户进行编译。



- ❖ 这里要求编译程序必须是可再入程序(reentry code)。
- ❖ 可再入程序具有这样的性质：它是纯代码，即在执行过程中自身不改变；可被多个程序同时调用的程序，调用它的程序应该提供各自独立的数据区。



进程的概念

- 是为了描述系统中各并发活动而引入的
- 进程是程序的一次执行。
 - ◆ 至少一个可执行程序
 - ◆ 一个独立的地址空间
 - ◆ 一个执行栈区（用于子程序调用，系统调用，进程切换）
 - ◆ 一些要使用的文件和I/O设备



进程具有的特性

- 动态性。进程是程序的一次执行过程，是临时的，有生命周期的。
- 独立性。进程是系统进行资源分配和调度的一个独立单位。
- 并发性。多个进程可在处理机上交替执行。
- 结构性。系统为每个进程建立一个进程控制块。



进程和程序

1. 进程是动态的，程序是静态的。程序是有序代码的集合，进程是程序的执行，没有程序就没有进程。通常，进程不可以在计算机之间迁移，而程序可以复制。
2. 进程是暂时的，程序是永久的。
3. 进程包括程序、数据、进程控制块。
4. 通过多次执行，一个程序可对应多个进程；通过调用关系，一个进程可包括多个程序。进程可创建其他进程，而程序并不能形成新的程序。



2.2 进程的描述

进程控制块 (PCB, process control block)

进程描述符(process descriptor)

- PCB是进程存在的唯一标识。
- 含有进程的描述信息和管理控制信息。
- 在UNIX中可通过进程文件系统(/proc)直接访问进程映像。



PCB中的基本信息

1. **进程标识数**: 用于唯一地标识一个进程，通常是一个整数。

外部标识符：由字母、数字所组成，由用户使用。如：send进程、print进程等。

2. **进程的状态、调度、存储器管理信息**: 是调度进程所必需的信息，包括进程状态、优先级、程序在主存地址、在外存的地址等。
3. **进程使用的资源信息**: 分配给进程的I/O设备、正在打开的文件等。

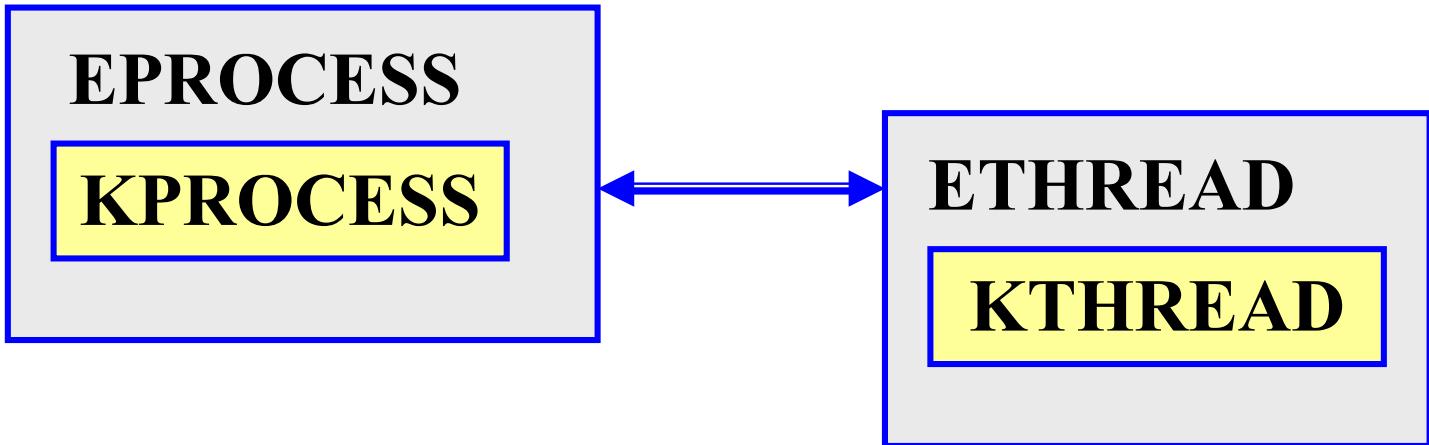


4. CPU现场保护区：保存进程运行的现场信息。
包括：程序计数器 (PC)、程序状态字、通用寄存器、堆栈指针等。
5. 记帐信息：包括使用CPU时间量、帐号等。
6. 进程之间的家族关系：类UNIX系统，进程之间存在着家族关系，父/子进程。Windows 进程之间不具有父子关系。
7. 进程的链接指针：链接相同状态的进程。



实例

- Unix: struct proc{.....};
Linux: struct task_struct{.....};
- Windows NT/2000/XP执行体进程块
(EPROCESS) 包含了管理地址空间的域、
跟踪分配给进程资源的域等等，基本信息同
Linux进程描述符。



内核实现线程调度，调度信息在KTHREAD结构中



进程空间

- 进程都有一个自己的地址空间，把该空间称为地址空间或虚空间。
- 进程空间的大小只与处理机的位数有关。
- 例如：一个16位长处理机的进程空间的大小为 2^{16} ；
- 一个32位长处理机的进程空间的大小为 2^{32} 。



程序的执行都是在进程空间内进行的。用户程序、进程的各种控制表格等都按一定的结构排列在进程空间中。

例：在**UNIX**与**Linux**等OS中，进程空间还被划分为用户空间、系统空间两部分：（如下图所示）



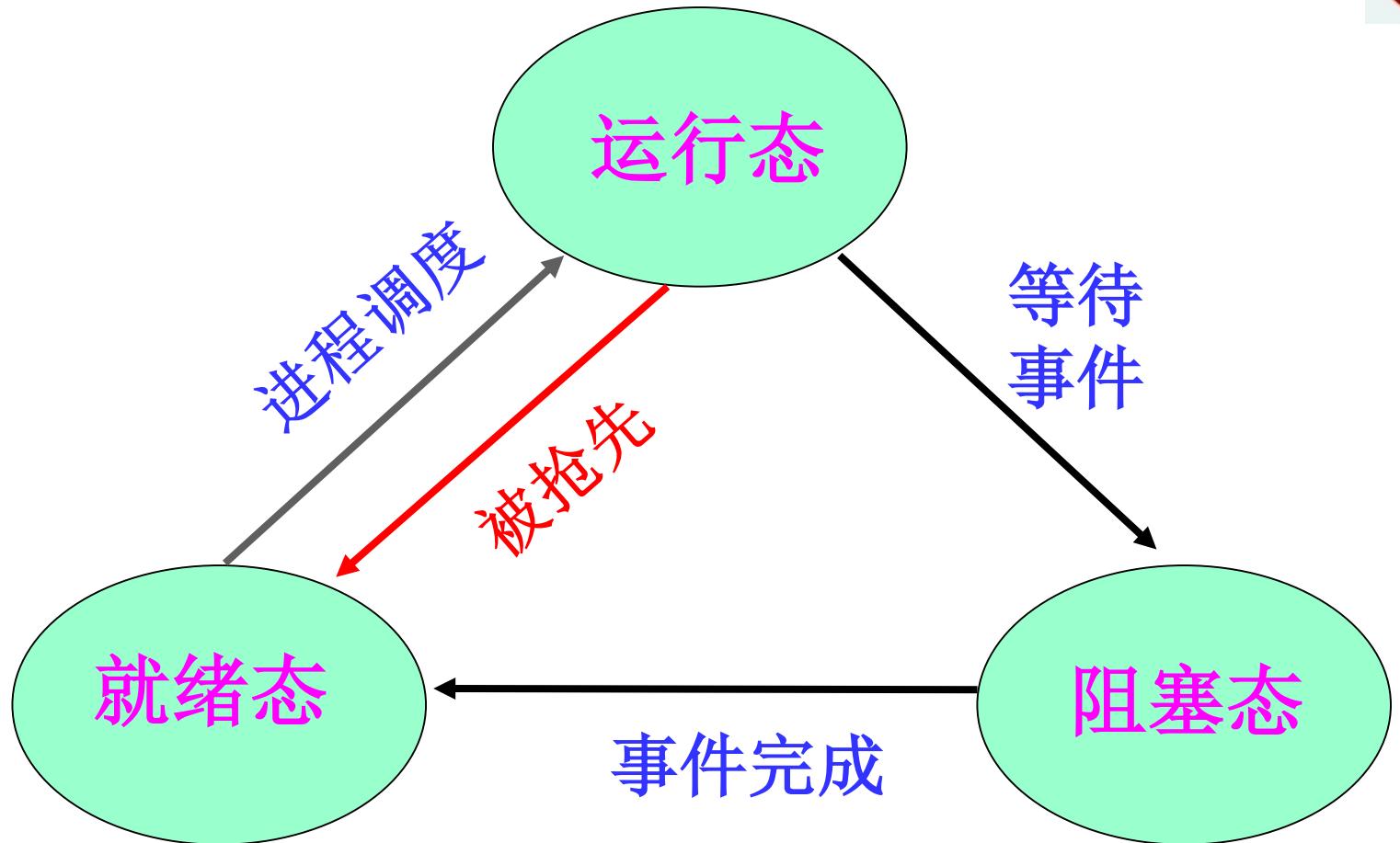


进程的状态

- 进程在其生命期内一直处在一个状态不断变化的过程中。为了刻画这变化过程，操作系统把进程分成若干种状态，并约定各种状态之间的转换条件。
- 状态信息记录在进程的**PCB**结构中。



- (1) **运行态** (running): 进程正在CPU上运行。
单CPU系统一次只有一个运行进程；多CPU系统可能有多个运行进程。
- (2) **阻塞态** (blocked): 又称**等待态**。当一个进程因等待某个条件发生而不能运行时所处的状态。等待I/O完成，等待一个消息
- (3) **就绪态** (ready): 已获得除CPU之外的全部资源，只要再获得CPU，就可执行。



进程的三个基本状态及其转换



由进程状态转换图可以看出

- **运行态—阻塞态：**是由运行进程自己主动改变的。**[例]**一个正在运行的进程启动了某一I/O设备后，使自己由运行态变为阻塞态，等待该I/O设备传输完成。
- **阻塞态—就绪态：**是由外界事件引起的。**[例]**当I/O设备传输完成时，请求中断，由I/O中断处理程序把因等待这一I/O完成而阻塞的进程变为就绪态。



■ 运行态—就绪态：处于运行态的进程被剥夺CPU。

[例] (1) 采用时间片轮转法调度：当前进程用完时间片，由运行态变为就绪态。

(2) 采用优先级调度：若有更高优先级的进程变为就绪态，当前进程被剥夺CPU，由运行态变为就绪态。

■ 就绪态—运行态：被进程调度程序选中。



- (4) **创建态**: 刚刚建立，未进就绪队列。
- (5) **终止态**: 已正常结束或故障中断，但尚未撤消。暂留在系统中，方便其它进程去收集该进程的有关信息。

创建态—就绪态: 操作系统准备好再接纳一个进程时，把一个进程从创建态变为就绪态。为了确保系统的性能，大多数系统都限制创建的进程数量。

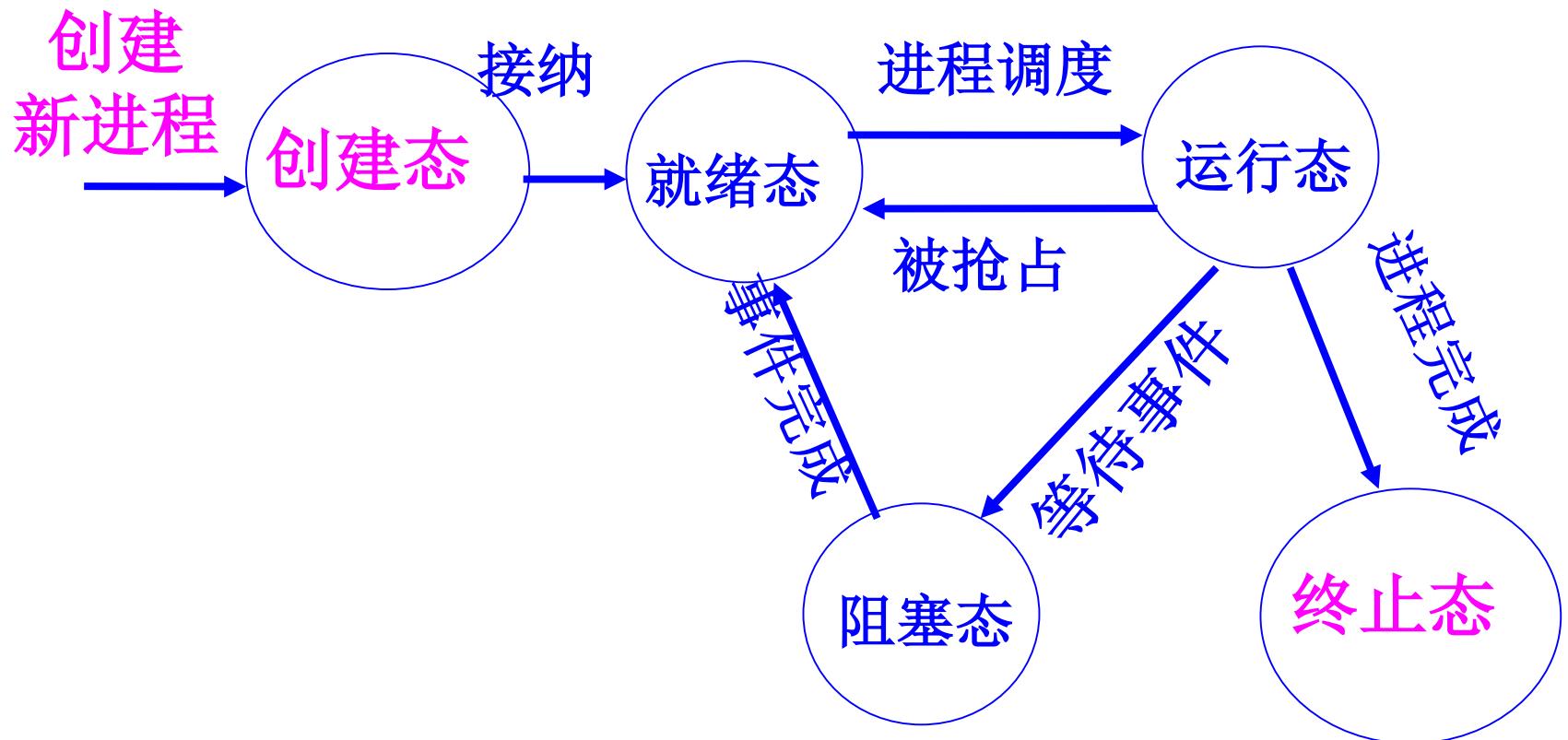


图2.3 进程的五种状态



进程的组织

(1) 线性表组织方式：把所有进程的PCB存放在一个数组中，系统通过数组下标访问每个PCB。

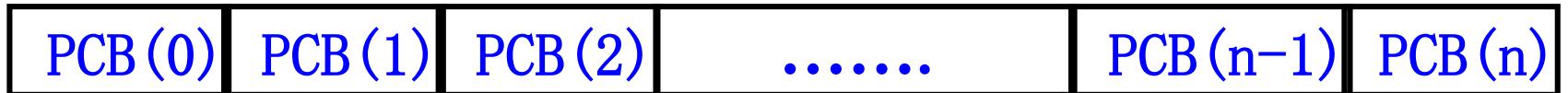
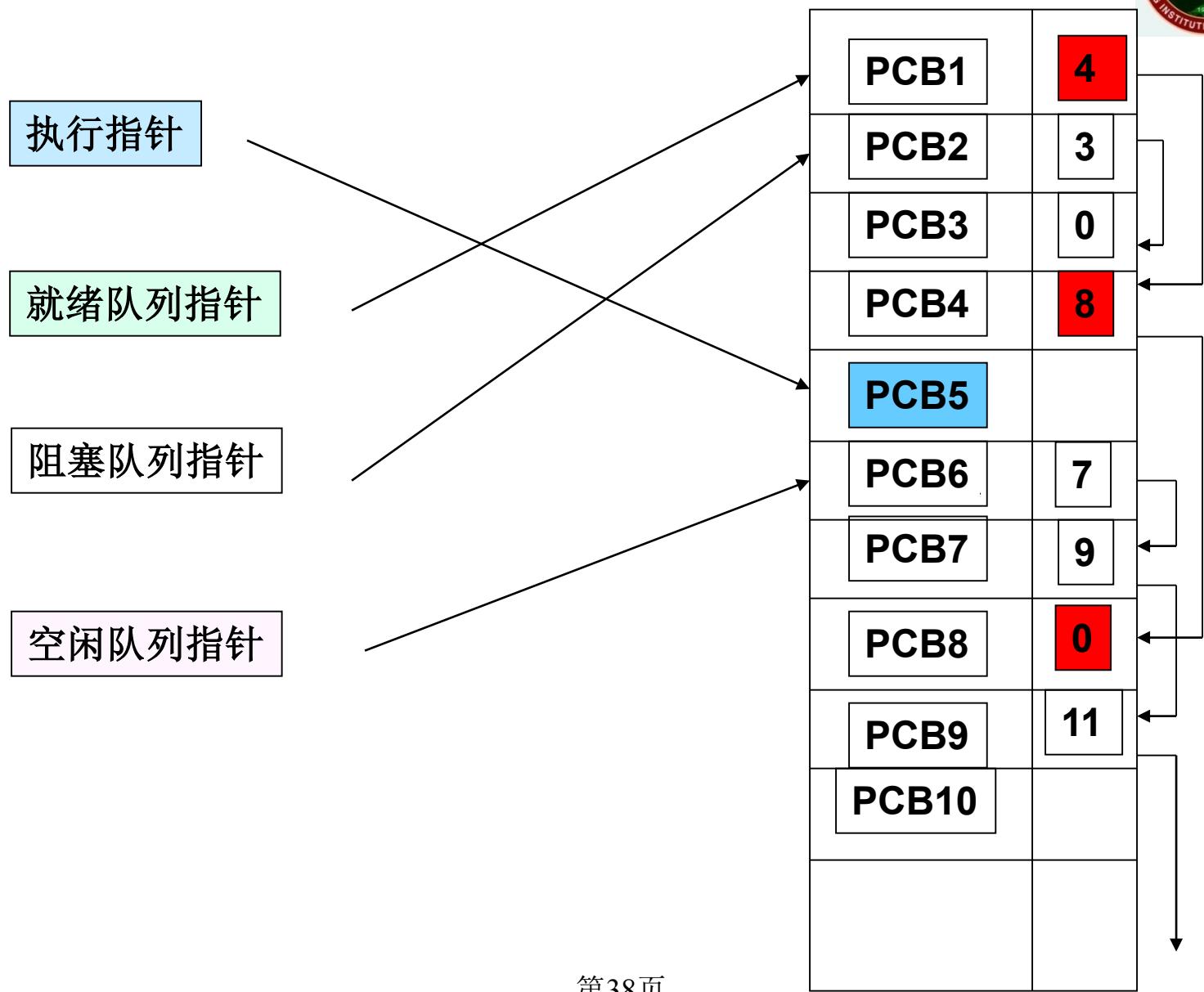


图2.4 进程控制块采用线性表管理



(2) 链表组织方式：把具有相同状态的PCB组成一个队列。

- ◆ 处于就绪态的进程可按照某种策略排成多个就绪队列。
- ◆ 处于阻塞态的进程又可以根据阻塞的原因不同组织成多个阻塞队列。如，等待磁盘I/O队列，等待磁带I/O队列等。

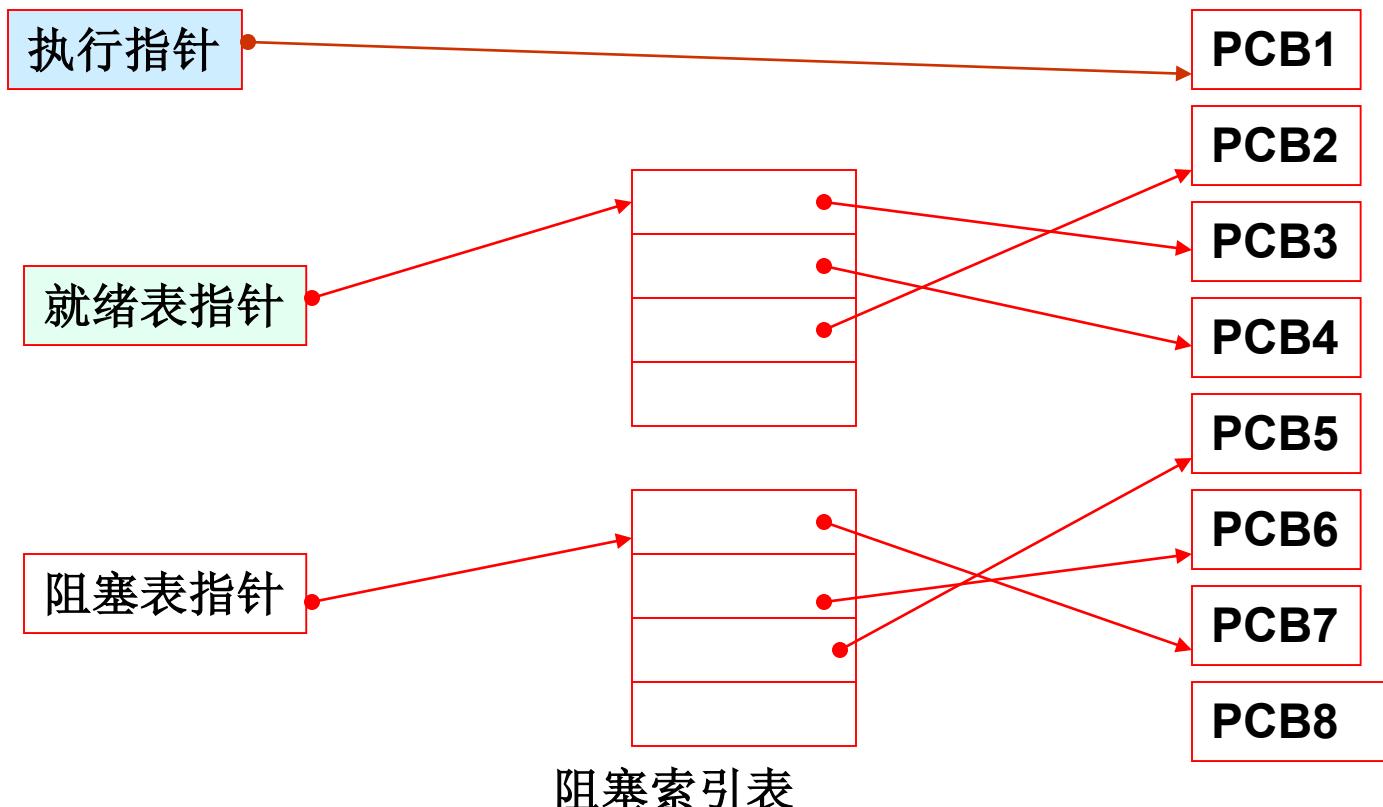


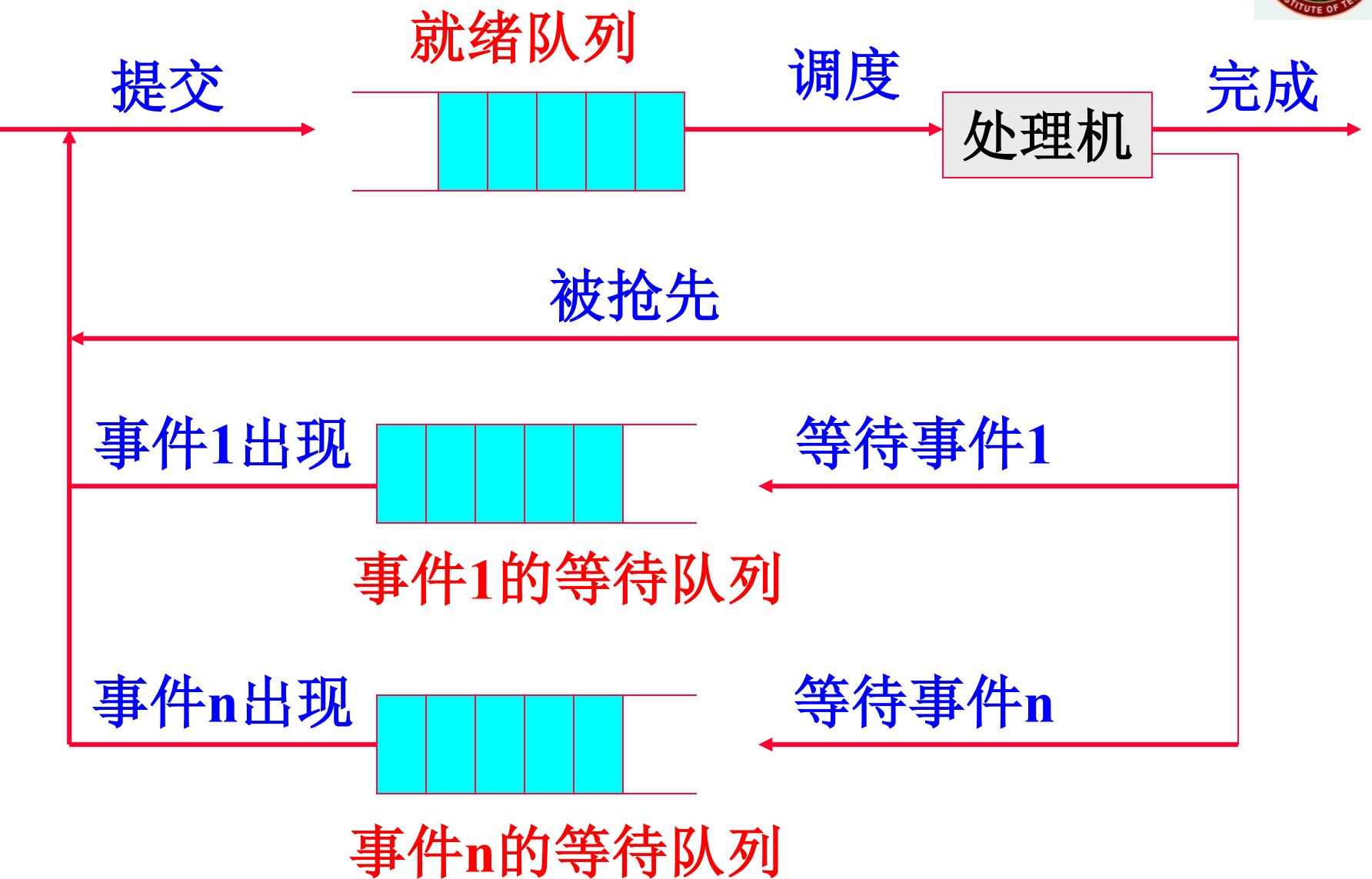


- 系统专门设置一个指针指向**当前运行进程的PCB**。
- UNIX系统中就有一个**CURPRO**指针，指向**当前运行进程的PCB**。



索引方式：系统根据进程的状态，建立几张索引表，并把索引表在内存的首地址记录于内存中一些专用单元。







2.3 进程控制

- 进程控制：是指系统使用一些具有特定功能的程序段来创建、撤销进程，以及完成进程各状态之间的转换。
- 进程控制是由操作系统内核实现的。
- 是属于原语一级的操作，不能被中断。



创建原语

(1) 创建进程的时机

- ① 系统初始化会创建若干进程
 - ◆ 前台进程同用户交互并替用户完成工作；
 - ◆ 后台守护进程(daemon)：接收电子邮件、接收对该机器中Web页面的访问请求、打印进程.....
- ② 在交互式系统中，键入一个命令或点击一个图标都会开始一个新的进程。



- ③ 在UNIX和Windows系统中，用户可以同时打开多个窗口，每个窗口都运行一个进程。
- ④ UNIX的系统调用`fork()`会创建一个与调用进程具有相同的副本的进程，子进程通过`execve`系统调用会运行一个新的程序。
- ⑤ Windows中的Win32函数调用`CreateProcess()`创建新进程，运行新程序。



(2) 进程创建的功能

- 扫描进程表，找到一个空闲的PCB。
- 为新进程的程序、数据、用户栈分配内存。
- 初始化PCB。把调用者提供的参数（进程名、进程优先级、实体所在主存的起始地址、所需的资源清单、记帐信息及进程家族关系等）填入PCB中。
- 将新进程插入就绪队列。



撤消原语

(UNIX中用exit(); Windows 用ExitProcess())

- 进程执行完或因故障不能继续运行。
- 功能：在PCB集合中寻找要撤销的进程；若有子孙进程，也须终止，以防成为不可控的；将其全部资源或归还其父进程或归还系统；撤销其PCB。



- 在**UNIX**系统中，子进程调用**exit()**后，处于**zombie**状态，进程管理器在父进程被告知子进程结束之前**不会释放其PCB**。在父进程执行了针对结束子进程的**wait()**系统调用后，处于**zombie**状态的子进程才真正结束。



阻塞原语

- 在运行过程中进程期待某一事件发生时，自己执行阻塞原语，由运行态变为阻塞态。
(等待键盘输入；等待磁盘数据传输完成；等待其它进程发送一个信息)
- 功能：中断CPU；将其运行现场保存在其PCB中；置状态为阻塞态，插入相应事件的阻塞队列中；转进程调度。
- Windows用**WaitForSingleObject()**。



唤醒原语

- 若进程等待的事件是I/O完成。I/O完成后，CPU响应中断，在中断处理中，将等待I/O完成而阻塞的进程唤醒，并置为就绪态。
- 若等待某进程发信息。由发送进程把该等待者唤醒，置为就绪态。插入就绪队列。



UNIX 阻塞/唤醒

1. Sleep()将在指定时间内阻塞本进程
2. Pause()阻塞本进程以等待信号。
3. Wait()阻塞本进程以等待子进程的结束。
4. Kill()向指定进程或进程组发送信号。
5. Wakeup()



挂起原语

- 实时系统，根据实时现场的需要，会将正在执行的或没有执行的进程挂起一段时间。被挂起的进程由活动状态变为静止状态（静止就绪、静止阻塞）。
- 分时系统，把进程从内存换到外存，进程就处于静止状态，不被调度。
- 调节系统负载：进程映像（进程地址空间内容、相关的内核数据、寄存器内容等）被交换到外存上。



解挂原语

- 当挂起进程的原因被解除时，系统调用解挂原语将指定的进程解挂，使其由静止状态变为活动状态。
- 当被解挂的进程变为**活动就绪**时，通常立即转进程调度。



2.4 处理机调度

- 进程数大于处理机数。多进程竞争处理机。
- 进程调度就是为进程分配处理机。
- 处理机调度涉及调度的策略、时机及进程的切换。
- 系统运行性能在很大程度上取决于调度。
吞吐量大小、周转时间长短、响应及时性。



处理机的调度级别

- **作业调度**: 高级调度。多道批处理系统。多个用户作业以成批的形式提交到外存，形成后备作业队列。被作业调度选中进内存，就处于运行态。
- **进程调度**: 低级调度。
- **交换调度**: 中级调度。将主存就绪或主存阻塞等暂不具备运行条件的进程换出到外存交换区；将外存交换区中的进程换入内存。交换调度可以控制进程对主存的使用。



- 进程调度要考虑CPU的利用率，因为进程切换的代价是比较高的。
 - ◆ 首先用户态必须切换到核心态；
 - ◆ 保存当前进程的状态以便以后重新装载
 - ◆ 通过运行调度算法选定一个新进程。



进程调度的功能

(1) 记录系统中各进程的执行状况

管理系统中各进程的控制块，将进程的状态变化及资源需求情况及时地记录到PCB中。

(2) 选择就绪进程真正占有CPU

(3) 进行进程上下文的切换

将正在执行进程的上下文保存在该进程的PCB中，将刚选中进程的运行现场恢复起来，以便执行。



进程上下文

为进程设置的相应的运行环境和物理实体

1. **用户级上下文**: 就是进程的程序和数据，用户栈。
2. **寄存器级上下文**: 是CPU的现场信息，包括程序计数器、PSW、栈指针、用来保存变量和临时结果的通用寄存器的值等。
3. **系统级上下文**: 包括进程的PCB、核心栈等。



- 栈记录进程的执行历程。栈帧中存放有关的输入参数、局部变量、过程调用完成之后的返回地址、没有保存在寄存器中的临时变量。
- 通常，每个进程会调用不同的过程，从而有一个各自不同的执行历程。



进程调度的方式

① 非抢先方式(非剥夺方式)

某一进程占用CPU, 直到运行完或不能运行为止, 其间不被剥夺。用在批处理系统。主要优点: 简单、系统开销小。

② 抢先方式(剥夺方式)

允许调度程序基于某种策略 (优先级、时间片等) 剥夺现行进程的CPU给其它进程。用在分时系统、实时系统。



进程调度的时机

- (1) 现行进程完成或错误终止；
- (2) 提出I/O请求，等待I/O完成时；
- (3) 在分时系统，按照时间片轮转，分给进程的时间片用完时；
- (4) 优先级调度，有更高优先级进程就绪；
- (5) 进程执行了某种操作原语，如阻塞原语和唤醒原语时，都可能引起进程调度。



进程调度算法

- 进程调度所采用的算法是与整个系统的设计目标相一致的。
 - ◆ 批处理系统：增加系统吞吐量和提高系统资源的利用率；
 - ◆ 分时系统：保证每个分时用户能容忍的响应时间。
 - ◆ 实时系统：保证对随机发生的外部事件做出实时响应。



(1) 先来先服务(FCFS): 简单，节省机器时间。缺点：容易被大作业垄断，使得平均周转时间延长。

[例] 几乎同时到达的三个作业j1、j2、j3。j1运行2小时，j2和j3只需1分钟。三个作业的平均周转时间为2个小时多。增长了短作业的周转时间。

(系统先运行j1，j2和j3要等2个小时。j1完成之后，j2和j3再分别运行1分钟。)



(2) 最短作业优先(SJF): 选取运行时间最短的作业运行。对短作业有利，作业的平均周转时间最佳。

若系统不断进入短作业，长作业就没有机会运行，出现**饥饿**现象。



(3) 响应比高者优先(HRN)

$R_p = (\text{作业等待时间} + \text{作业估计运行时间}) / \text{作业估计运行时间}$

$$= 1 + \text{作业等待时间} / \text{作业估计运行时间}$$

- 特点：结合了先来先服务、短作业优先的方法。优先运行短作业和等待时间足够长的长作业。
- 缺点：算法比较复杂。



(4) 优先级调度法

- 将CPU分配给就绪队列中优先级最高的进程。
- 静态优先级：在进程创建时确定的，运行时保持不变。通常赋予系统进程较高优先级；申请资源量少的赋予较高优先级。可能导致低优先级的长进程没有机会运行。
- 动态优先级：原优先级可随进程的推进而改变。根据进程占用CPU时间的长短或等待CPU时间的长短动态调整。



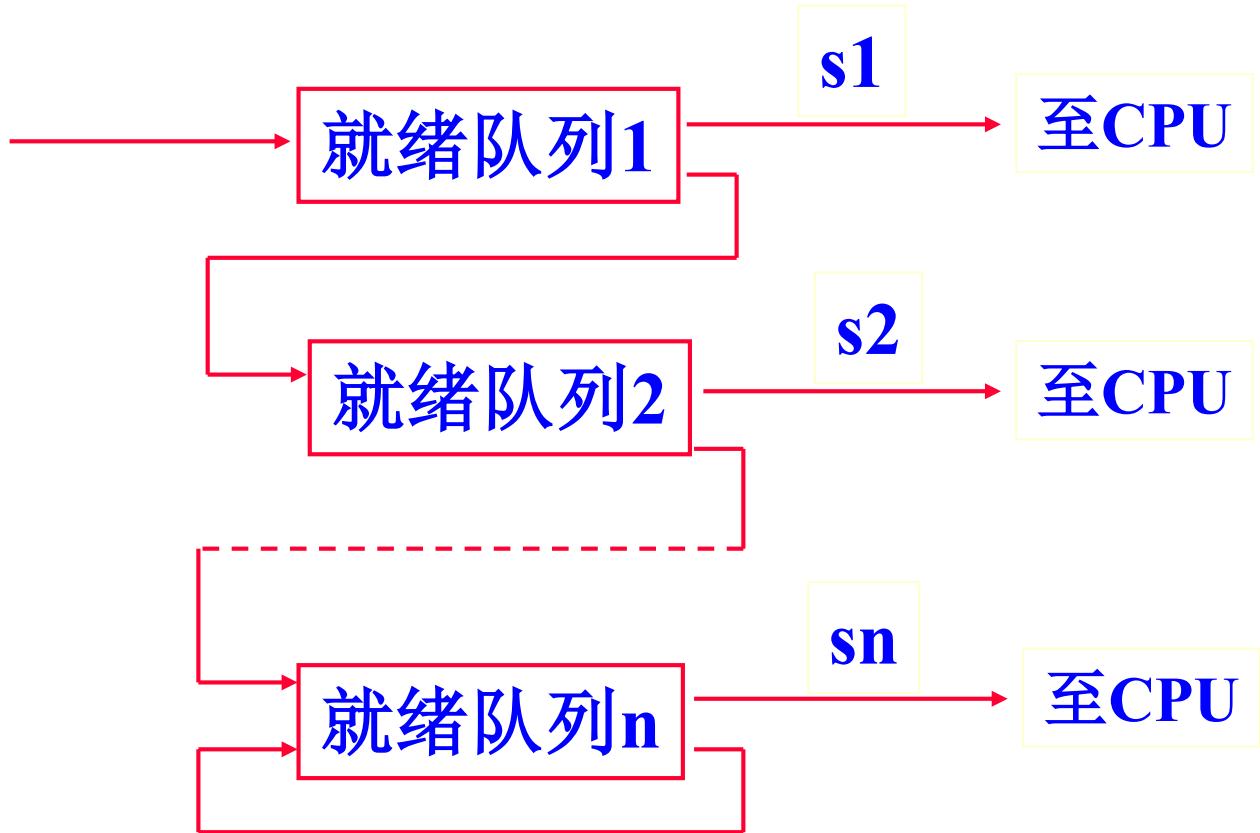
(5) 轮转法(Round Robin)

- 用在分时系统，轮流调度所有就绪进程。
- 利用一个定时时钟，使之定时地发出中断。时钟中断处理程序在设置新的时钟常量后，立即转入进程调度程序。
- 时间片长短的确定原则：既要保证系统各个用户进程及时地得到响应，又不要因时间片太短而增加调度的开销，降低效率。



(6) 多级反馈队列轮转法

- 因就绪原因不同，系统通常设置多个就绪队列。多个就绪队列可采用前后台运行。
- 前台队列采用RR调度；后台采用FCFS。
- 高优先级进程的时间片较短，低则较长。
- 刚创建的进程和因请求I/O而未用完时间片的进程排在高优先级队列。
- 运行2~3个时间片还未完成的进程降级。



(通常，时间片： $s1 < s2 < s3 < \dots < sn$)



实时系统的调度算法

- **时钟驱动法：**各任务的调度安排通常是在系统运行前就确定了。硬实时系统的任务是固定的和可知的。系统以规则的间隔时间调度任务执行。一个定时器被周期性地设置，时间到期后，系统启动要执行的任务。
- **加权轮转法：**进程的权就是分配给它的一小部分处理机时间。轮转时，不同的进程可以获得不同的处理机时间。广泛用在高速开关网的实时控制中。



多处理器调度

- 几乎所有的现代操作系统都支持对称多处理器的系统架构。
- 多处理器调度面临多处理器负载均衡和处理器亲和性问题。
- 负载均衡可以更好地提高多个处理器的利用率。
- 处理器亲和性：调度程序避免进程切换，尽量使其在同一个处理器上运行，以提高缓存内数据的命中率。



2.5 线程的引入

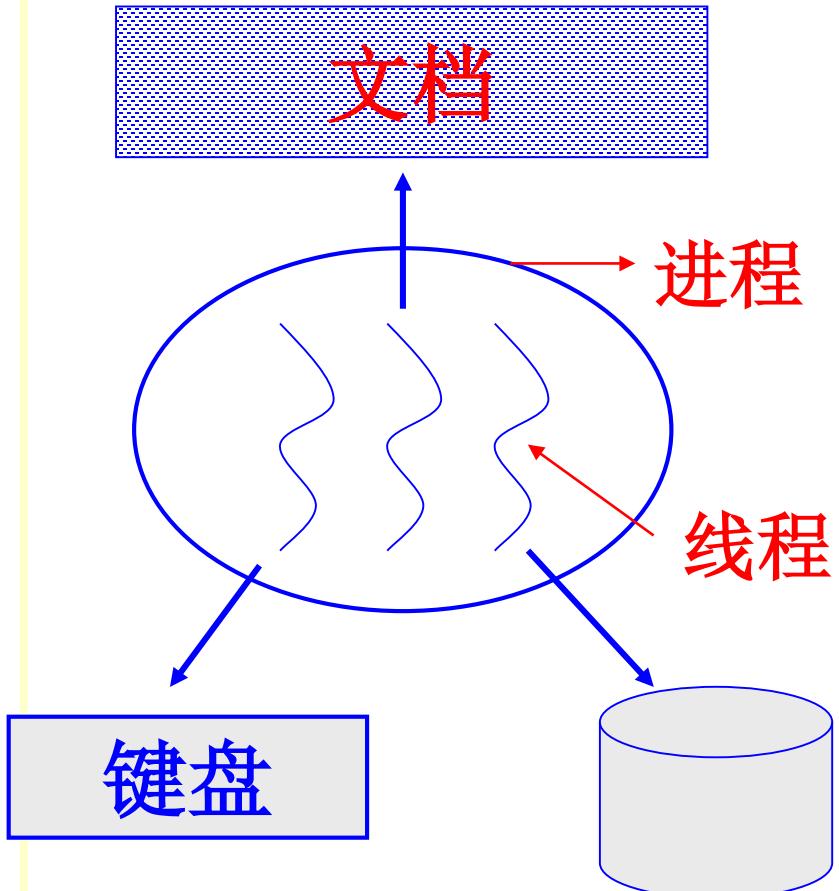
- 传统操作系统，每个进程有一个地址空间和一个控制线程。
- 需要多线程的主要原因：在许多应用中同时发生着多种活动，其中某些活动随着时间的推进会被阻塞。可以将这些应用分解成多个线程（开销小），并发运行（性能的考虑）。



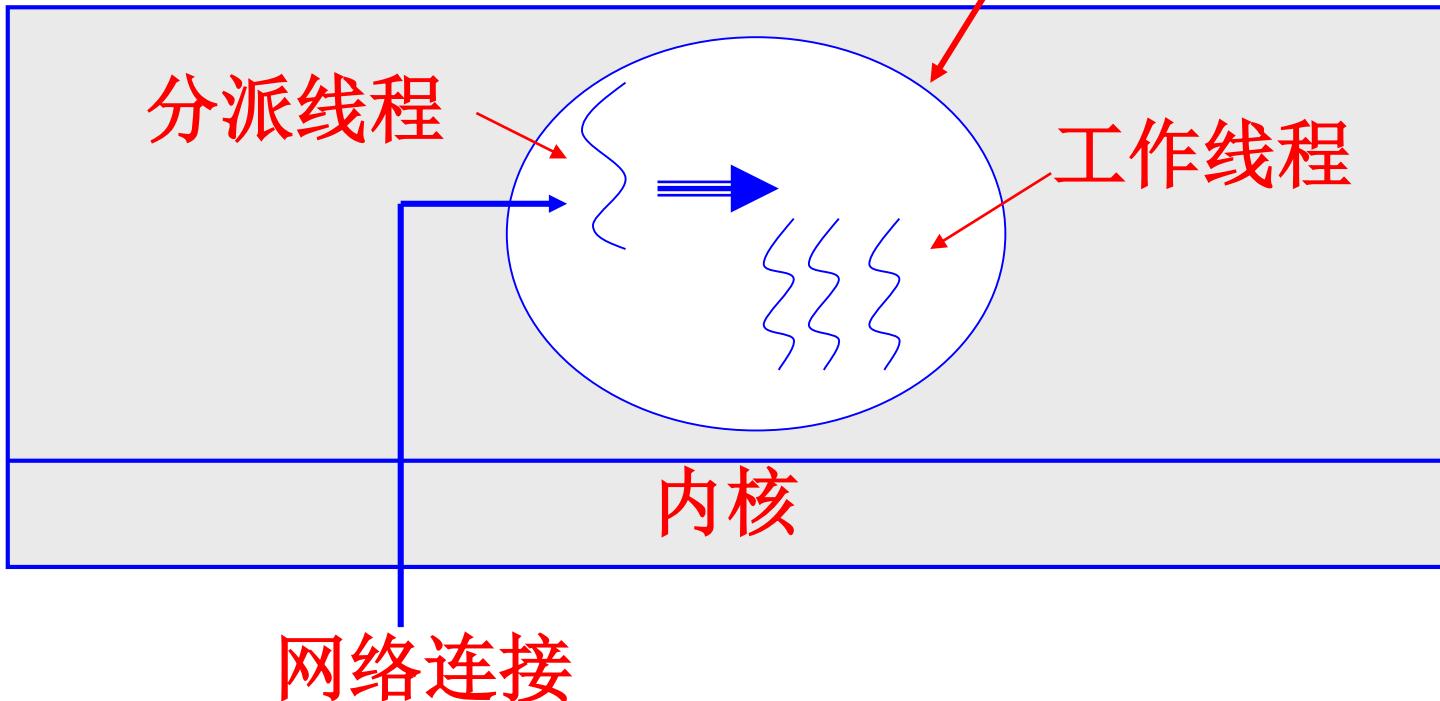
- 进程的基本属性：资源的拥有者和处理机调度的基本单位。
- 引入线程后，进程还是资源的拥有者，但线程继承了进程的处理机调度属性。
- 线程成为了一个执行实体（轻型进程）。

多线程字处理：

- ◆ 一个与用户交互线程
- ◆ 另一个在后台进行格式化的线程。
- ◆ 一旦在第一页中的语句被删除掉，交互线程就立即通知格式化线程对整本书重新进行处理。
- ◆ 同时，交互线程继续监控键盘和鼠标。
- ◆ 第三个线程可以处理磁盘备份。



用户空间



- 分派线程(dispatcher)从网络读入请求，之后选一个工作线程提交请求，当该工作线程阻塞在磁盘操作上时，分派线程可选另一个工作线程运行。



- 进程内的多线程共享同一个地址空间和所有可用数据。由于线程附带的资源不多，所以线程比进程更容易创建和撤销。
- 线程(thread)：是进程内的一个可执行实体，是处理机调度的基本单位。



进程的多线程模型

程序段

数据段

...

打开文件

进程
控制块

线程1

线程2

线程3

寄存器组

寄存器组

寄存器组

堆栈

堆栈

堆栈

线程
控制块

线程
控制块

线程
控制块

进程地
址空间



一个线程的组成

- 有一个唯一的标识符
- 表示处理机状态和运行现场的一组寄存器
- 两个堆栈，分别用于用户态和核心态调用时进行参数传递
- 一个独立的程序计数器
- 关联的进程和线程指针



线程与进程的比较

(1) 拥有的资源

- 进程拥有一个独立的地址空间，若干代码段和数据段，若干打开文件、主存以及至少一个线程。
- 一个进程内的多线程**共享该进程的所有资源**，线程自己拥有很少资源。

(2) 调度

- 进程调度需进行进程上下文的切换，开销大。
- 同一进程内的线程切换，仅把线程拥有的一小部分资源变换了即可，效率高。同一进程内的线程切换比进程切换快得多。**不同进程的线程切换...**



(3) 并发性

引入线程后，使得系统的并发执行程度更高。进程之间、进程内的多线程之间可并发执行。

(4) 安全性

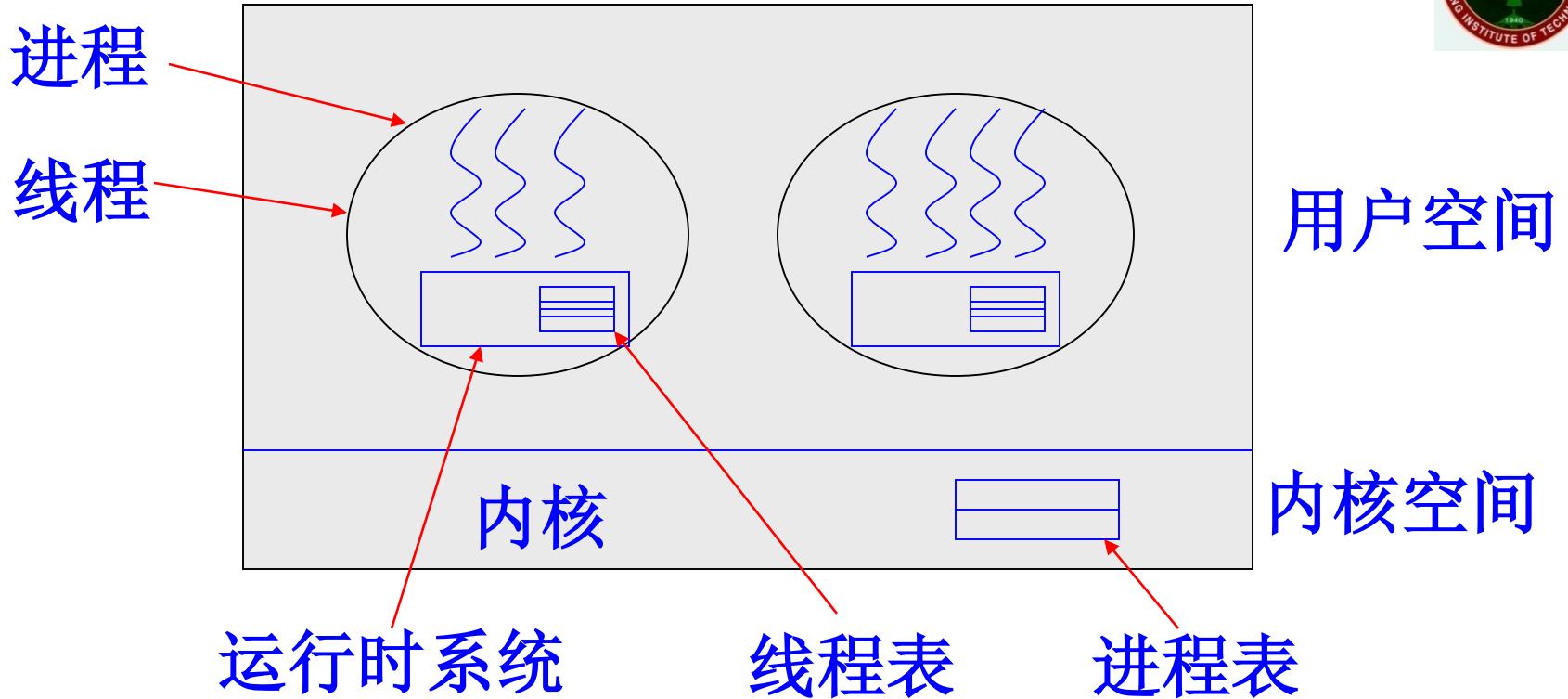
同一进程的多线程共享进程的所有资源，一个线程可以改变另一个线程的数据，而多进程实现则不会产生此问题。共享方便。



系统对线程的支持

1) 用户级线程

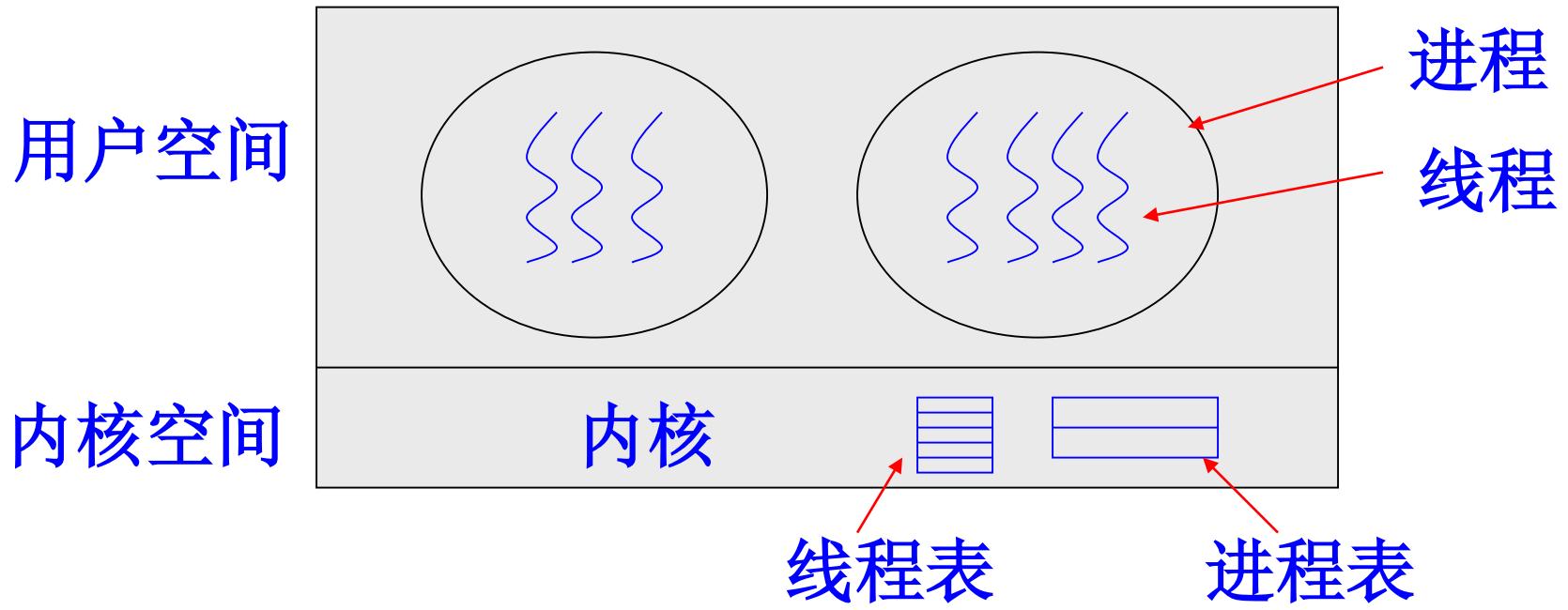
- ◆ 有关线程的所有管理工作都由用户进程通过调用线程库完成。自己设计线程调度算法。
- ◆ 内核以进程为单位进行调度。一个线程阻塞，其依附的进程也阻塞。
- ◆ 多线程对应核心级一个进程。
- ◆ 如，POSIX的Pthread线程库



运行时系统（Run-time system）是一个管理线程的过程集合，包括：`thread_create`、`thread_exit`、`thread_wait`。

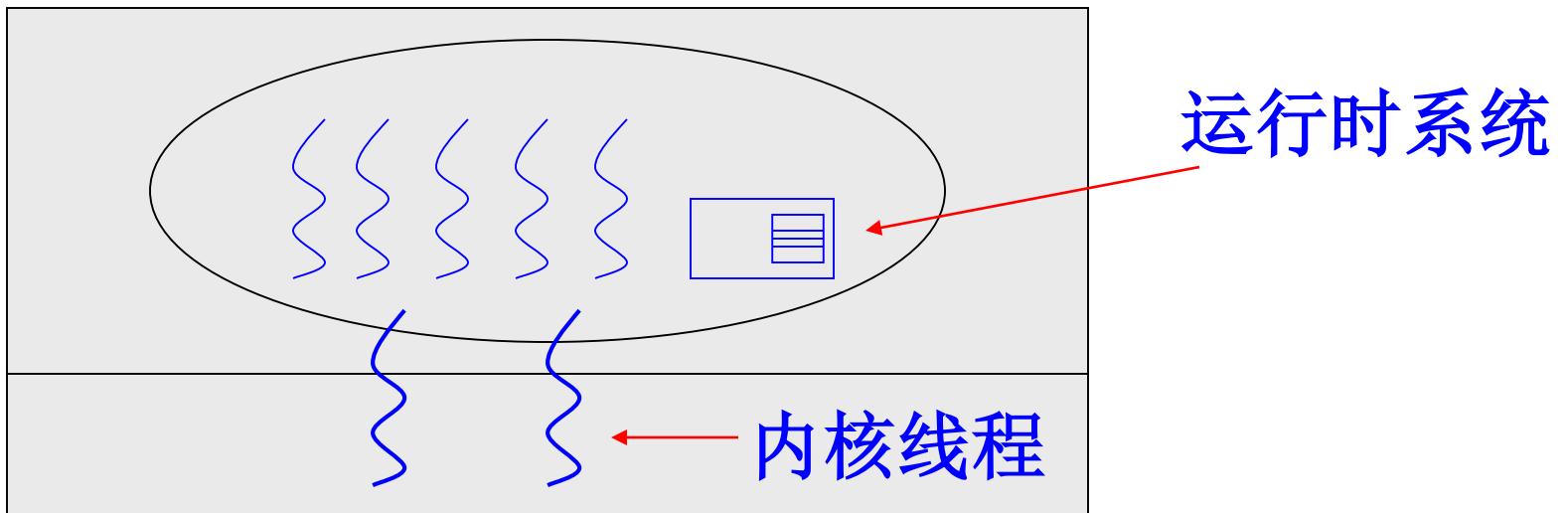
2) 核心级线程

- 有关线程的管理工作都由内核完成。应用程序通过系统调用来创建或撤销线程。
- 一个线程的阻塞，不影响其他线程的执行。
- Windows Linux 多处理器系统



3) 两级组合

- 既支持用户级线程，也支持核心级线程。
- 用户级多个线程对应核心级多个线程。
- 当内核了解到一个线程阻塞后，通知运行时系统，重新调度其他线程。



Solaris 用户线程 \leftrightarrow LWP \leftrightarrow 内核线程
第83页



1. 由于线程拥有较少的资源，又具有传统进程的许多特性，因此有时把线程叫做轻型进程。把传统的进程叫做重型进程。
 - ◆ **Linux**, 线程就是轻量级进程(**LightWeight Process**)。一个进程拥有一组共享其地址空间和资源的轻量级进程。Clone()函数
2. 创建进程时，系统同时为进程创建第一个线程。进程中的其它线程是通过调用线程创建原语显式创建的。
 - ◆ **Windows**, 创建线程的函数
CreateThread(...,lpStartAddress,...)