

第三章 进程描述与控制





§ 1 为什么要引入进程

一、前趋图

前趋图是描述程序执行先后顺序的有向无循环图：

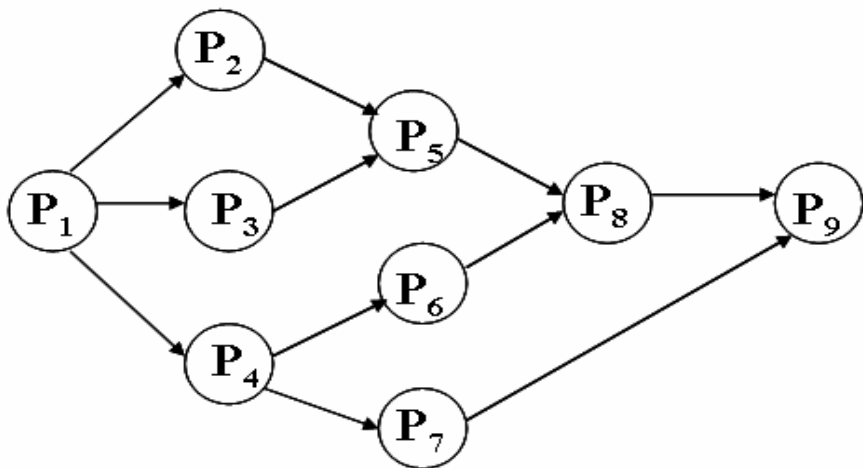
1. 结点 P_i ：可以表示一条语句、一段程序或一个进程。

2. 前驱(偏序)关系 \rightarrow

$\rightarrow = \{ (P_i, P_j) \mid P_i \text{ must complete before } P_j \text{ may start} \}$

若 $(P_i, P_j) \in \rightarrow$ ，则称 P_i 是 P_j 的直接前驱，而 P_j 是 P_i 的直接后继。

3. 没有前趋的结点称初始结点，没有后继的结点称作终止结点。

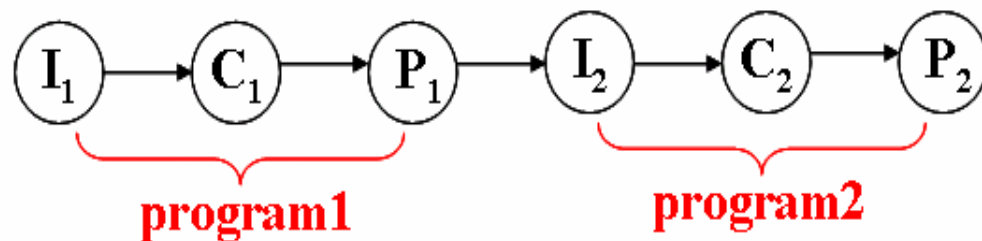
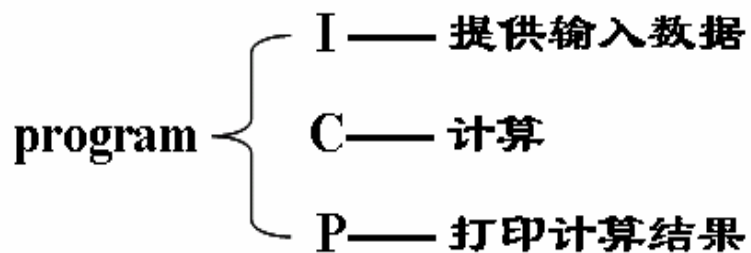

$$P = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9\}$$
$$\rightarrow = \{ (P_1, P_2), (P_1, P_3), (P_1, P_4), (P_2, P_5), (P_3, P_5), (P_4, P_6), (P_4, P_7), (P_5, P_8), (P_6, P_8), (P_7, P_9), (P_8, P_9) \}$$



§ 1 为什么要引入进程

二、程序顺序执行

例:有一组计算程序以单道方式顺序执行



特征

顺序性: 程序执行的过程是按程序规定的逻辑连续执行的过程, 上一指令结束是下一指令开始的充要条件

封闭性: 程序执行得到的结果只由程序自身决定, 不受外界因素包括其他程序的影响

确定性: 程序执行结果与执行速度无关, 程序以任何速度多次执行, 只要初始条件相同, 结果唯一可再现

静态的程序和动态的程序执行一一对应

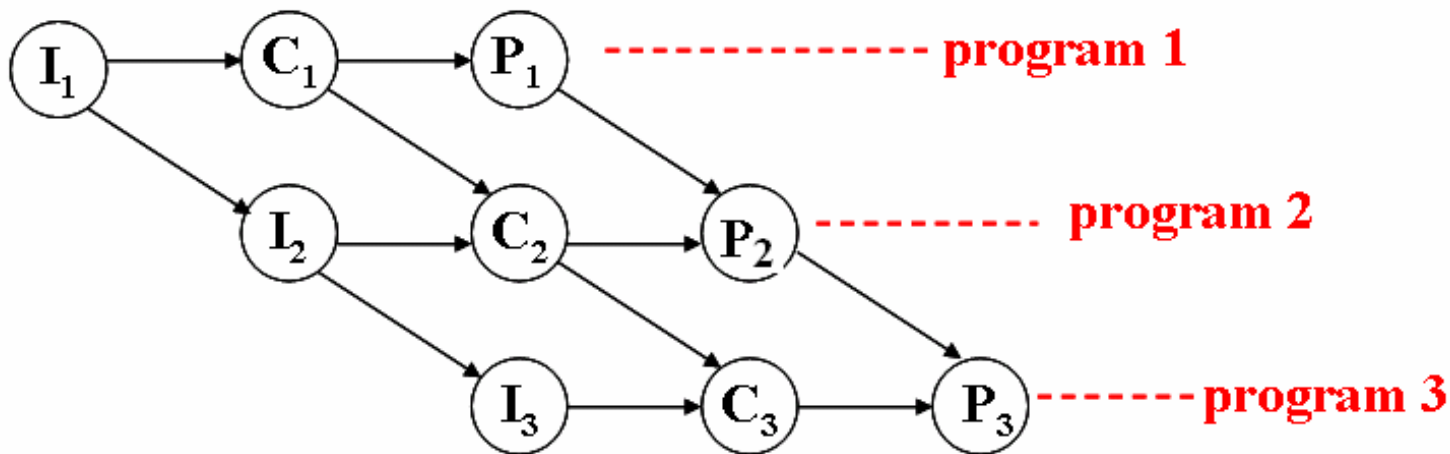


§ 1 为什么要引入进程

三、程序并发执行

例:有一组计算程序以多道方式并发执行

program {
 I — 提供输入数据
 C — 计算
 P — 打印计算结果





§ 1 为什么要引入进程

三、程序并发执行



中断性：程序的执行失去连续性，呈现间断性特征

特征

例：有 A、B 程序并发执行并共享变量 N。形式化描述如下：

```
var N:integer:=0;
```

```
begin
```

```
  parbegin
```

```
    program A: begin
```

```
      L1: N:=N+1;
```

```
      goto L1;
```

```
    end
```

```
    program B: begin
```

```
      L2: print N;
```

```
      N:=0;
```

```
      goto L2;
```

```
    end
```


```
  parend
```


```
end
```



§ 1 为什么要引入进程

三、程序并发执行

 **中断性**：程序的执行失去连续性，呈现中断性特征

特征  **失去封闭性**：程序执行结果与速度有关，呈现**不可再现性**

 **程序和程序的执行活动不一一对应**

可再入程序

在执行过程中不改变自身代码的程序，也称为纯代码程序

- 具有可再入性，是指能被多个程序同时调用而被共享
- 各调用者自行提供工作区，以收容可变部分



§ 1 为什么要引入进程

四、Bernstein条件

1.读集： $R(P_i)=\{a_1, a_2, \dots, a_m\}$ ，表示 P_i 执行需参考变量集。

2.写集： $W(P_i)=\{b_1, b_2, \dots, b_n\}$ ，表示 P_i 的执行要改变的变量集。

3.Bernstein条件：

若程序 $P1$ ， $P2$ 满足下述条件便能并发执行并获得正确结果。

$$R(P1) \cap W(P2) \cup R(P2) \cap W(P1) \cup W(P1) \cap W(P2) = \{ \quad \}$$

eg: $s_1: c=a+b$; $s_2: d=c+1$;

则: $R(s_1) = \{a, b\}$ $R(s_2) = \{c\}$

$$W(s_1) = \{c\} \quad W(s_2) = \{d\}$$

$$\because R(s_1) \cap W(s_2) \cup R(s_2) \cap W(s_1) \cup W(s_1) \cap W(s_2) \neq \{ \quad \}$$

$\therefore s1$ 和 $s2$ 之间存在一个前趋（偏序），表示为： $\textcircled{s_1} \rightarrow \textcircled{s_2}$



§ 1 为什么要引入进程

四、Bernstein条件

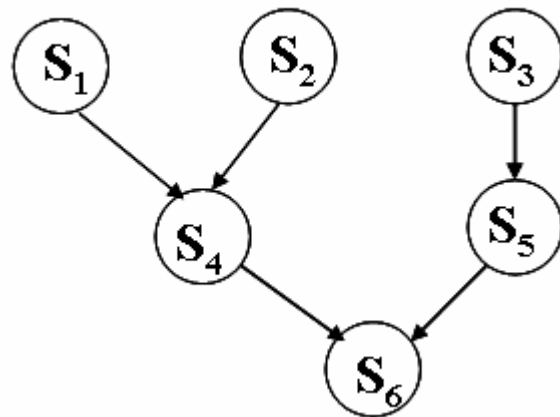
例1：根据Bernstein条件，则在如下4条语句中：

S1: $a:=x+y$ S2: $b:=z+1$ S3: $c:=a-b$ S4: $w:=c+1$

S1和S2两条语句可以并发执行，S3和S4两条语句不可以并发执行。（回答本小题应考虑：是否可以并发执行）

例2：已知求值公式 $(A^2+3B)/(B+5A)$ ，若A、B已赋值，试画出该公式求值过程的前趋图。

分析： S1: $x1=A*A$ S2: $x2=3*B$
S3: $x3=5*A$ S4: $x4=x1+x2$
S5: $x5=B+x3$ S6: $x6=x4/x5$





§ 2 什么是进程

一、定义和特征

- **Dijkstra**: 程序在处理机上**执行时**发生的谓之进程
- **Donovan**: 进程是可以和别的计算**并行**执行的计算
- **A.Lan.C.shaw**: 进程是**程序**与其**数据**顺序通过处理机所发生的
- **E.Cohen**: 进程是系统进行资源分配和调度的一个**独立**单位
- **78年庐山会议**: 进程是具有**独立**功能的**程序**关于某个**数据集**的一次**运行**活动



动态性

进程 特征

进程具有生命期: 因创建而产生、调度而执行、撤销而消亡。在生命期中历经一系列离散状态及其变迁





§ 2 什么是进程

一、定义和特征

- **Dijkstra**: 程序在处理机上**执行时**发生的谓之进程
- **Donovan**: 进程是可以和别的计算**并行**执行的计算
- **A.Lan.C.shaw**: 进程是**程序**与其**数据**顺序通过处理机所发生的
- **E.Cohen**: 进程是系统进行资源分配和调度的一个**独立**单位
- **78年庐山会议**: 进程是具有**独立**功能的**程序**关于某个**数据集**的一次**运行**活动

动态性

并发性: 一组进程在执行时间上具有重迭, 有并发区

并发性

顺序性: 一个进程只呈现一条控制线索, 是顺序的

结论: 进程间并发粒度粗糙 (引入线程)


进程
特征



§ 2 什么是进程

一、定义和特征

- **Dijkstra:** 程序在处理机上**执行时**发生的谓之进程
- **Donovan:** 进程是可以和别的计算**并行**执行的计算
- **A Lan.C.shaw:** 进程是**程序**与其**数据**顺序通过处理机所发生的
- **E.Cohen:** 进程是系统进行资源分配和调度的一个**独立**单位
- **78年庐山会议:** 进程是具有**独立**功能的**程序**关于某个**数据集**的一次**运行**活动



动态性



并发性



独立性

进程特征

独立调度:进程拥有独立的CPU现场, 切换时现场保护与恢复, 切换开销大

独立拥有资源:

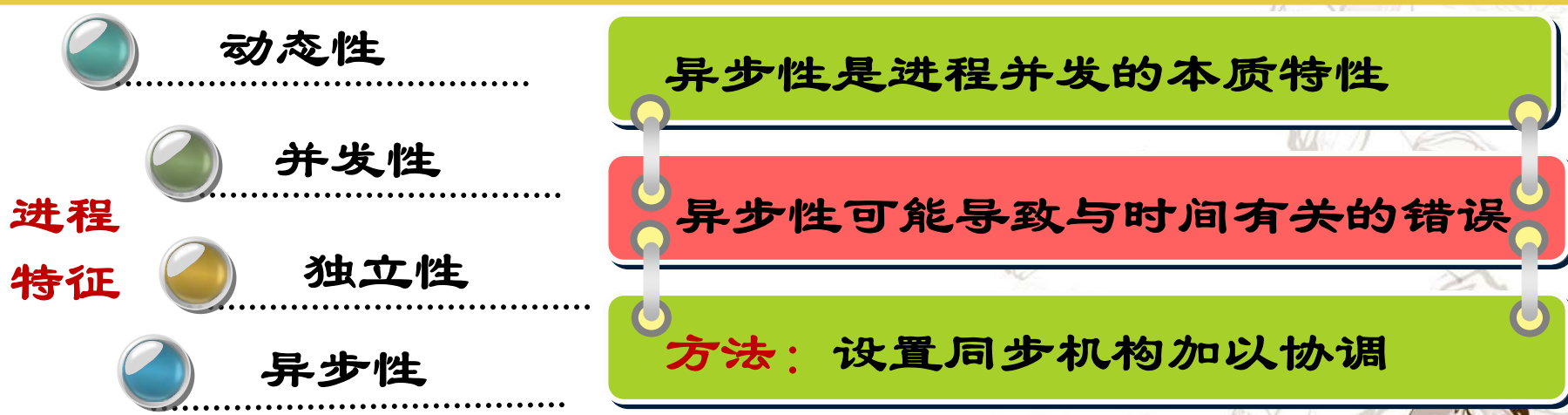
- 进程创建、撤消等管理开销大
- 进程虚地址空间彼此独立、关系疏远, 除非采用进程通信手段



§ 2 什么是进程

一、定义和特征

- **Dijkstra:** 程序在处理机上**执行时**发生的谓之进程
- **Donovan:** 进程是可以和别的计算**并行**执行的计算
- **A Lan.C.shaw:** 进程是**程序**与其**数据**顺序通过处理机所发生的
- **E.Cohen:** 进程是系统进行资源分配和调度的一个**独立**单位
- **78年庐山会议:** 进程是具有**独立**功能的**程序**关于某个**数据集**的一次**运行**活动





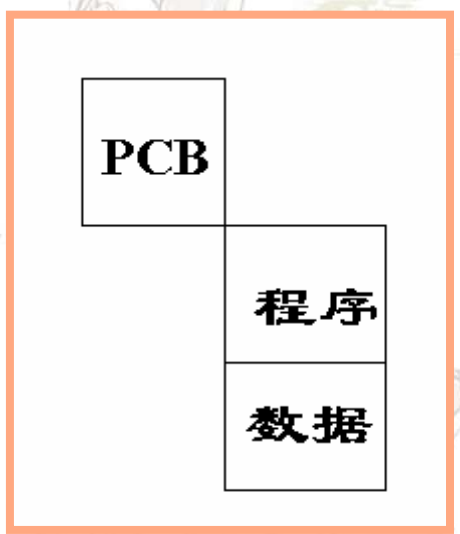
§ 2 什么是进程

一、定义和特征

- **Dijkstra:** 程序在处理机上**执行时**发生的谓之进程
- **Donovan:** 进程是可以和别的计算**并行**执行的计算
- **A Lan.C.shaw:** 进程是**程序**与其**数据**顺序通过处理机所发生的
- **E.Cohen:** 进程是系统进行资源分配和调度的一个**独立**单位
- **78年庐山会议:** 进程是具有**独立**功能的**程序**关于某个**数据集**的一次**运行**活动

- 进程特征
- 动态性
 - 并发性
 - 独立性
 - 异步性
 - 结构性

从静态逻辑结构上,
进程由**程序**、**数据**
和**进程控制块PCB**三
部分组成,统称作**进
程映像或进程上下文**





§ 2 什么是进程

例1：在操作系统中，B 是竞争和分配资源的基本单位。
A. 程序 B. 进程 C. 作业 D. 用户

例2进程与程序的主要区别在于进程是动态的，
而程序是静态的。一个程序可对应多个进程。

例3：进程与程序的本质区别是D。
A. 存储在内存和外存 B. 顺序和非顺序执行机器指令
C. 分时和独占使用计算机 D. 动态和静态特征

例4：在单处理机系统中实现并发技术后，C。
A. 各进程在某一个时刻并行运行，CPU与外设间并行工作
B. 各进程在某一个时间段并行运行，CPU与外设间串行工作
C. 各进程在某一个时间段并行运行，CPU与外设间并行工作
D. 各进程在某一个时刻并行运行，CPU与外设间串行工作



§ 2 什么是进程

二、进程控制块

PCB是存放进程的管理和控制信息的数据结构。在创建进程时建立，在撤消进程时消亡，伴随进程运行的全过程，是进程存在的唯一标志。





§ 2 什么是进程

三、UNIX 进程结构

- **proc结构**：进程基本控制块。用以存放无论进程是否在CPU上执行，核心都需要查询和修改的信息，必须常驻主存。如：进程标识符、进程状态、优先级、指向user结构的指针等
- **user结构**：进程扩充控制块。存放进程不在CPU上执行，核心无需查询和修改的信息，不必常驻主存。如：指向proc结构的指针、用户打开文件表、系统调用返回值等
- **进程上下文**：进程实体及其进程运行所需的支撑环境合称进程上下文，其中环境指各类寄存器、页表、核心数据结构等。进程上下文是进程执行活动全过程的静态描述。
- **UNIX进程定义**：进程的执行是进程在其上下文中的执行。更确切地说，是在它当前上下文层中的执行。



§ 2 什么是进程

例1：是非题

- 1.进程是提交给计算机系统的用户程序。(**F**)
- 2.当进程处于非执行状态时，其PCB可以被全部交换到磁盘上。(**F**)
- 3.PCB是进程存在的唯一标识。(**T**)

例2：单处理机系统中，可并行的是 **D** 。

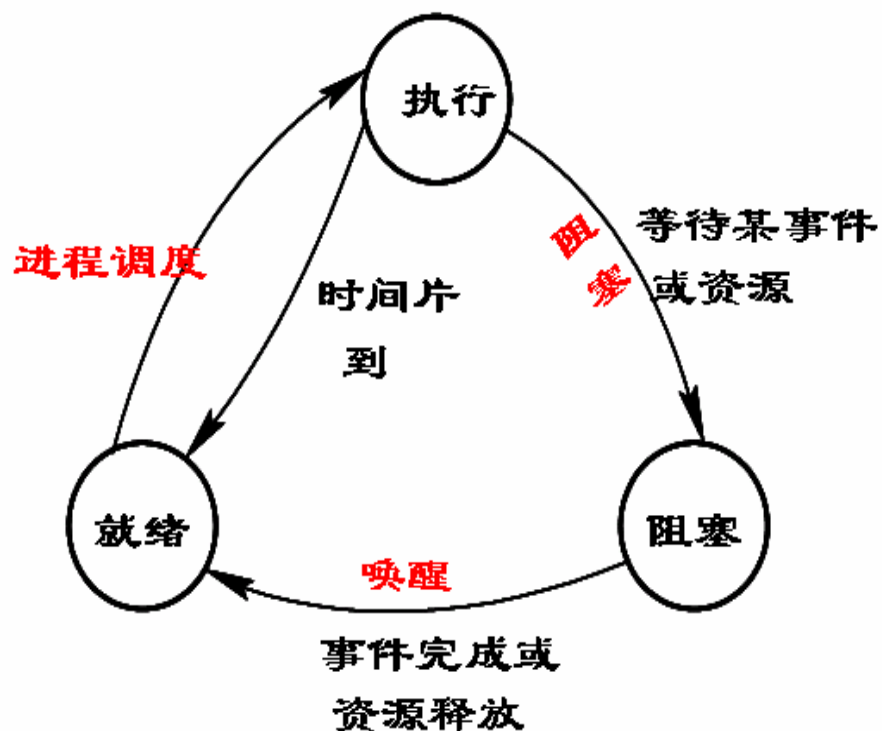
I 进程和进程 II 处理机与设备 III 处理机与通道 IV 设备与设备

A. I、II和III B.I、II和IV C.I、III和IV D.II、III和IV



§ 3 进程状态及转换

一、进程基本三态



1.就绪态(Ready)

进程已获得除CPU之外的一切所需资源，一旦获得CPU就可以执行的状态

就绪队列RL:

——就绪进程PCB所构成的链表

2.执行态(Running)

进程占据CPU向前执行的状态

3.阻塞态(Blocked)、睡眠态

进程因等待某事件发生而暂停运行的状态，**是进程自行阻塞**

阻塞队列WL:

——按不同阻塞原因链接形成的PCB链



§ 3 进程状态及转换

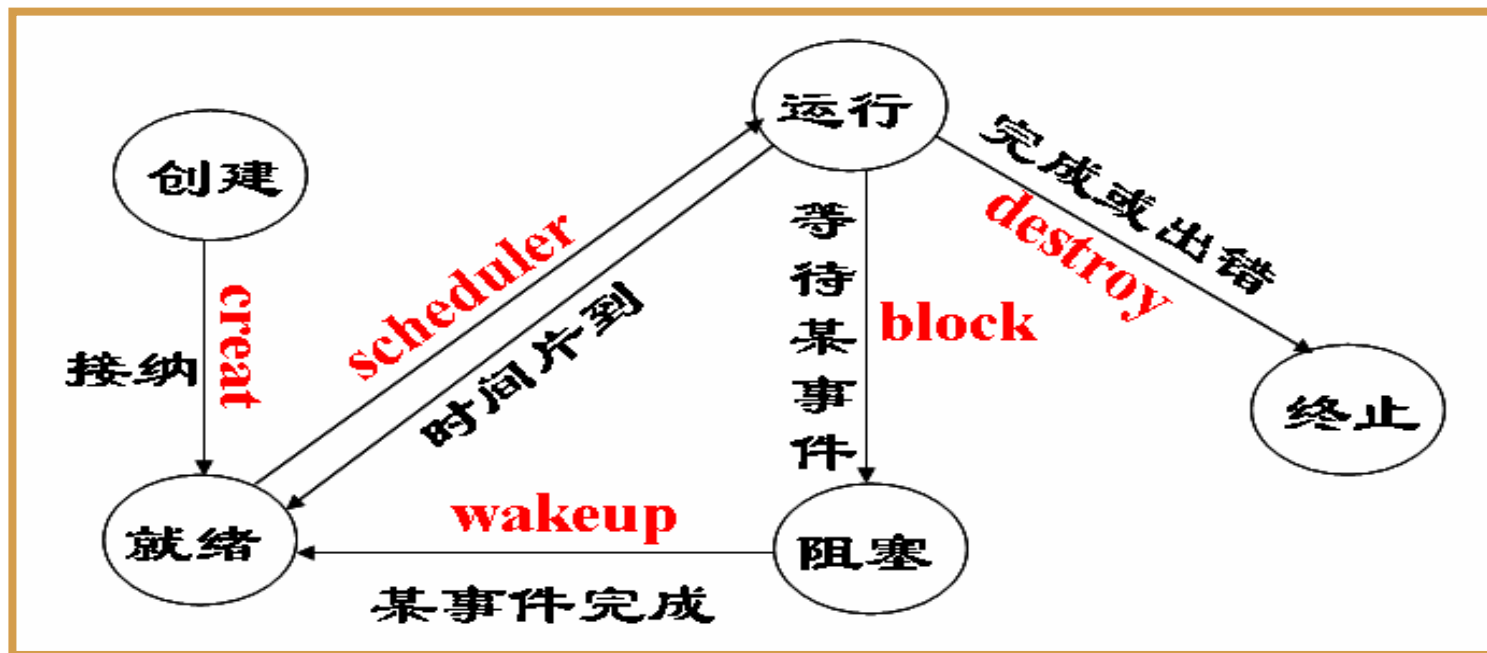
二、进程五态模型

创建态(New)

进程正被创建,未被接纳进入RL的状态
(创建PCB, 审核并分配必要资源)

终止态(Exit)

进程运行结束或出错,正回收其占有的
资源且供父进程收集相关信息的状态





§ 3 进程状态及转换

例1：下列进程状态变化中，A 是不可能直接发生的。

- A. 等待->运行
- B. 等待->就绪
- C. 运行->就绪
- D. 运行->等待

例2：进程由就绪态转换为运行态是由C引起的。

- A. 中断事件
- B. 进程状态转换
- C. 进程调度
- D. 为程序创建进程

例3：分时系统中，一个进程完成打印，将导致另一个等待打印机的进程的状态 B。

- A. 从阻塞到运行
- B. 从阻塞到就绪
- C. 从就绪到运行
- D. 从就绪到阻塞



§ 3 进程状态及转换

例4：在一个单处理机系统中，若有5个用户进程，且假设当前时刻为用户态，则处于就绪状态的用户进程最多有 4 个，最少有 0 个。

例5：一个单处理机的系统中有 n 个用户进程，在不考虑进程状态过渡的情况下，运行进程的个数为 $0\sim 1$ 个，就绪进程的个数为 $0\sim n-1$ 个，阻塞进程的个数为 $0\sim n$ 个。





§ 3 进程状态及转换

三、进程挂起操作

- **改善CPU利用率:**挂起阻塞进程以便就绪进程进入内存
- **调节系统负荷:**挂起不紧迫进程以平滑负载峰值
- **考察进程:**终端用户挂起指定进程以便检查
- **进程同步:**父进程使用挂起协调各子进程推进速度
- **OS需要:**检查资源使用、记账、系统故障时挂起某些进程

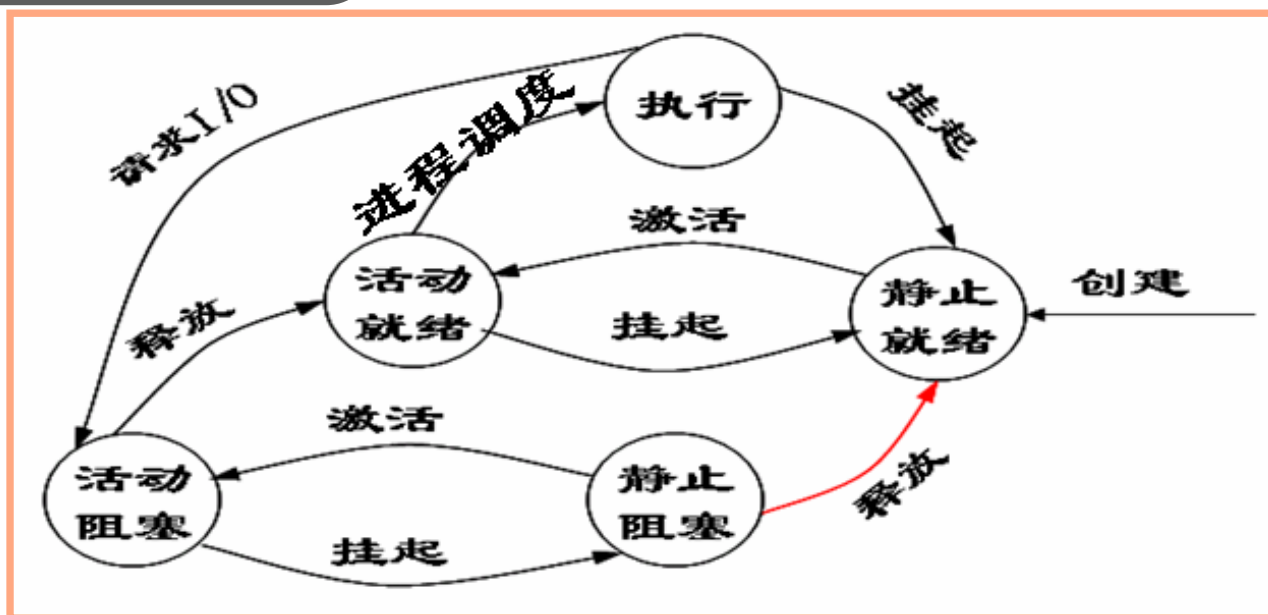
● **挂起(suspend):**当发生引起进程挂起的事件时, 将指定进程挂起。挂起状态的进程, 意味着不占用内存空间, 其进程映像的非常驻部分被对换至磁盘上。被挂起进程处于静止状态

● **激活(active):**从挂起断点启动,使进程重处活跃态、从外存转入内存的操作



§ 3 进程状态及转换

三、进程挂起操作



- **挂起(suspend)**: 当发生引起进程挂起的事件时, 将指定进程挂起。挂起状态的进程, 意味着不占用内存空间, 其进程映像的非常驻部分被对换至磁盘上。被挂起进程处于静止状态
- **激活(active)**: 从挂起断点启动, 使进程重处活跃态、从外存转入内存的操作



§ 3 进程状态及转换

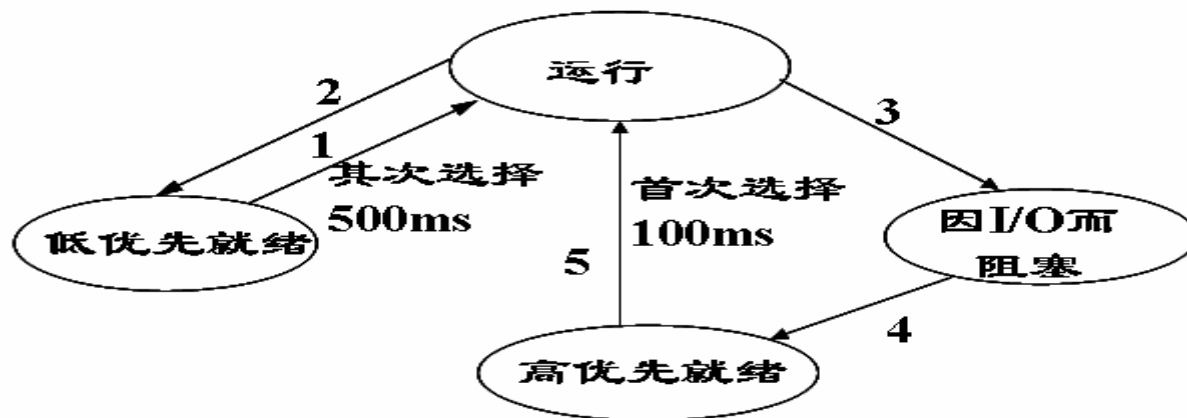
例1:当怀疑一进程的中间结果时,应对其予以 C。

A.阻塞 B.撤消 C.挂起 D.激活

例2:某系统进程状态变迁如图所示,且假设系统的进程调度方式是可剥夺方式。(1)说明进程发生变迁1、变迁3、变迁5的原因。(2)当发生一个变迁可能引起另一个变迁的发生,则这两个变迁称为因果变迁。下述因果变迁是否会发生,如果可能,会在什么情况下发生?

a. 3->5 b. 3->2 c. 2->1 d. 4->1 e. 4->5

(3)根据此状态变迁图说明该系统的调度策略、调度效果。



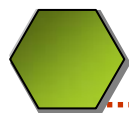


§ 4 进程控制(交通切换)

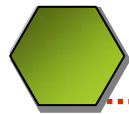
负责创建/撤消进程及进程的状态变迁

一、操作系统内核

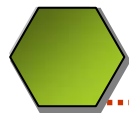
将OS中**与硬件紧密相关模块**、**运行频率高的功能模块**以及**公用操作模块**，安排在靠近硬件的OS低层次中，在初启(自举)操作系统时加载它们进入并常驻主存，即OS Kernel。



与硬件紧密相关: eg. 中断处理程序、设备驱动程序



运行频率高的: eg. 进程调度、进程控制



公用操作模块: eg. 进出链表、进出栈



§ 4 进程控制(交通切换)

二、primitive

定义:原语是由若干指令构成,用以完成特定系统功能的程序

是不可分割的原子操作:其操作要么不做,要么全做

通过屏蔽中断保证其原子操作特性

例:辨析**Primitive** & **System call**的区别与联系

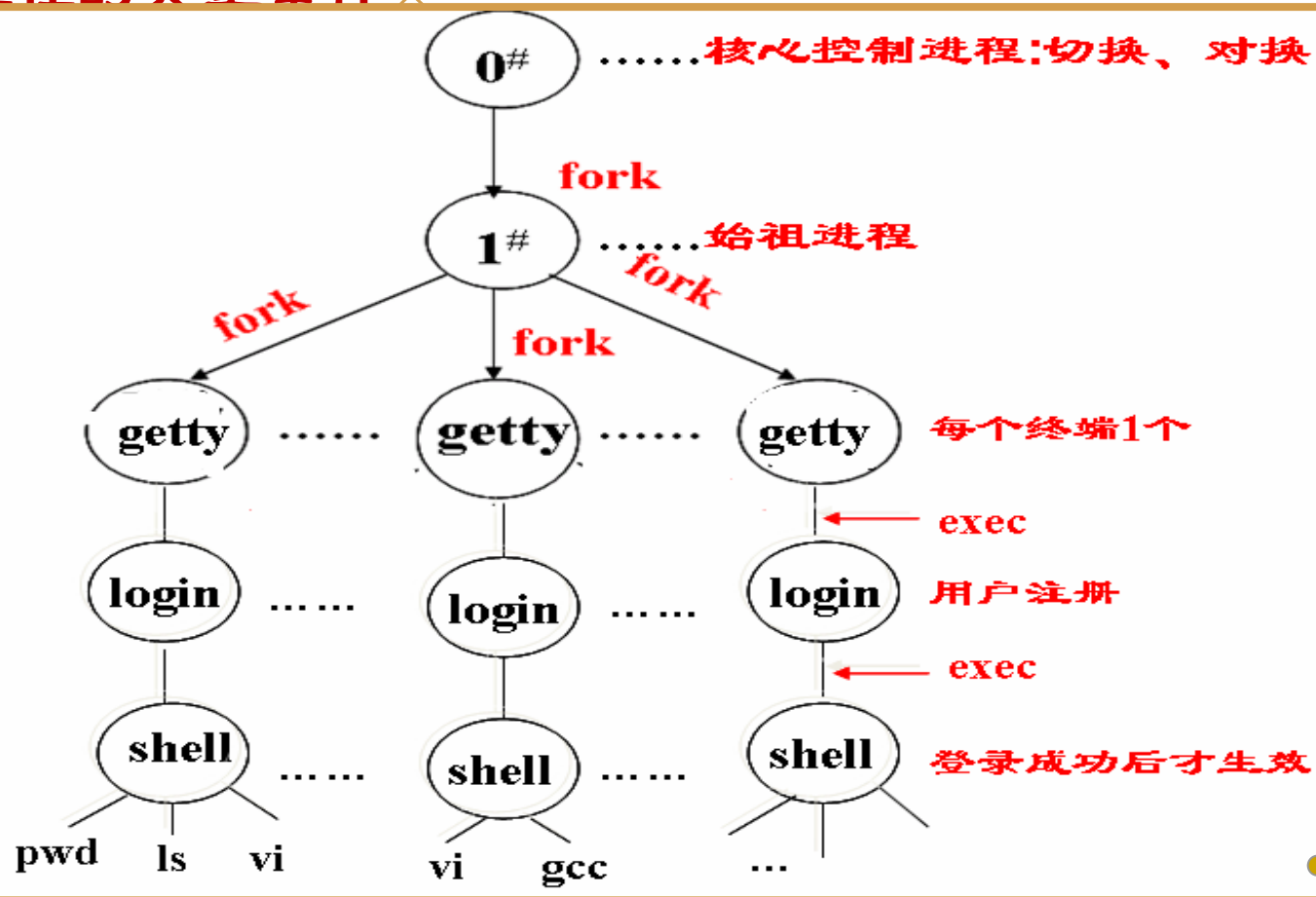
- 系统调用的执行允许被中断,原语是不可分割的原子操作。
- 系统调用的实现可能使用了原语,甚至使用了多个原语。
- 有些系统调用本身就是原语,但系统调用并不都是原语。而原语是一种特殊的系统调用。



§ 4 进程控制(交通切换)

三、创建原语

※创建进程的典型事件※



UNIX自举及进程家族树生成

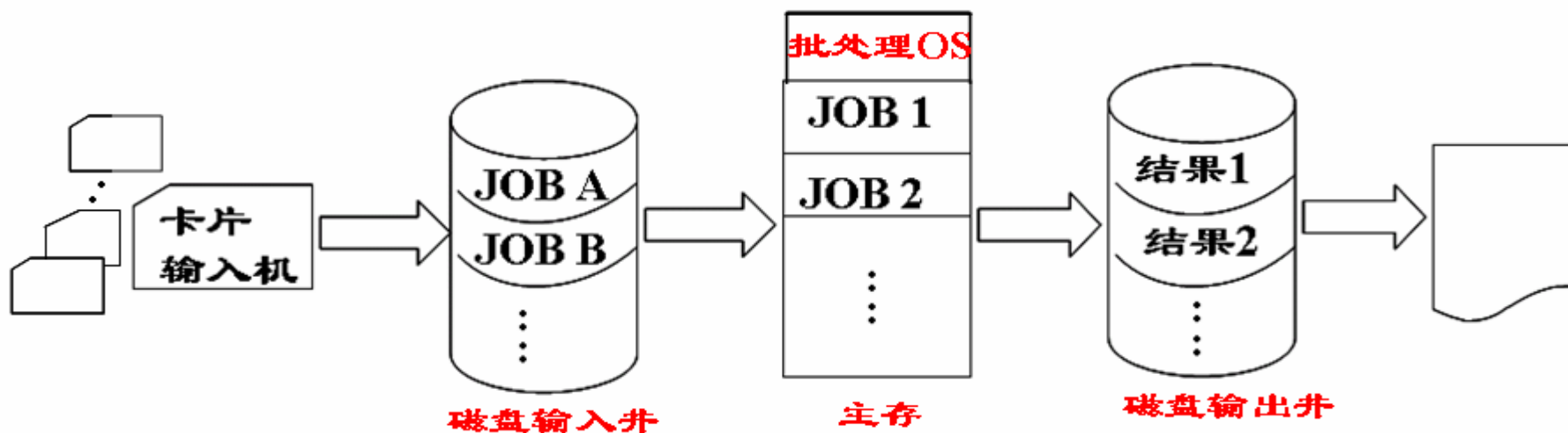


§ 4 进程控制(交通切换)

三、创建原语

※创建进程的典型事件※

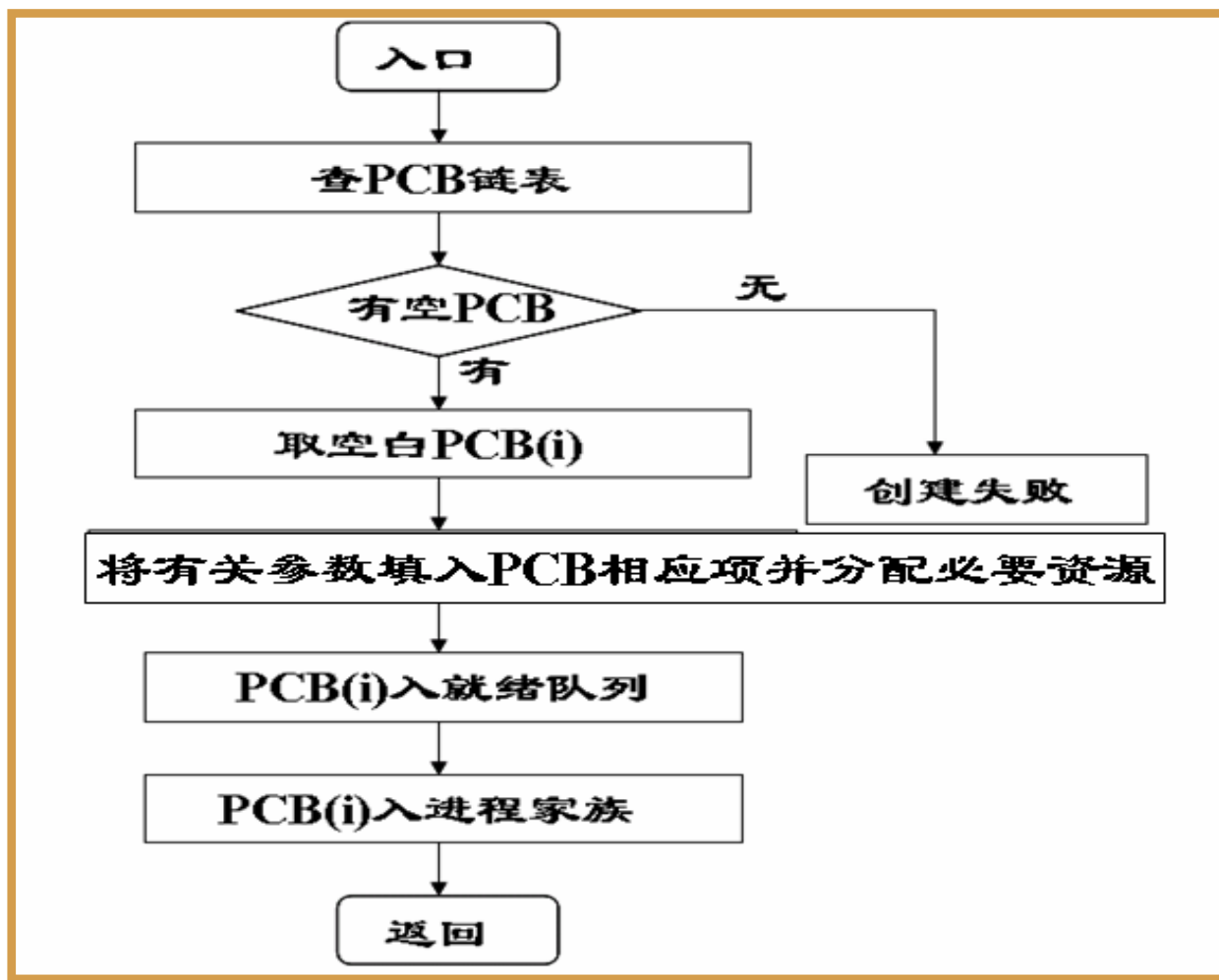
- **分时系统**：**用户登录**为用户人机会话创建终端会话进程。
 - **批处理系统**：**作业调度**为选中的作业创建1或多个进程。
 - **提供服务**：某些OS通过创建服务进程，为用户提供服务。
- UNIX等系统是将进程由用户态运行变迁至核心态运行。
- **应用请求**：用户态进程根据应用需求创建子进程。





§ 4 进程控制(交通切换)

三、创建原语





§ 4 进程控制(交通切换)

四、撤销原语

当进程**完成**任务
或遇到**异常**无法
继续运行时，终止
该进程，有利于
及时回收进程
占用的资源

正常结束:进程任务已完成，常在其程序文本最后安排一条导致进程消亡的指令

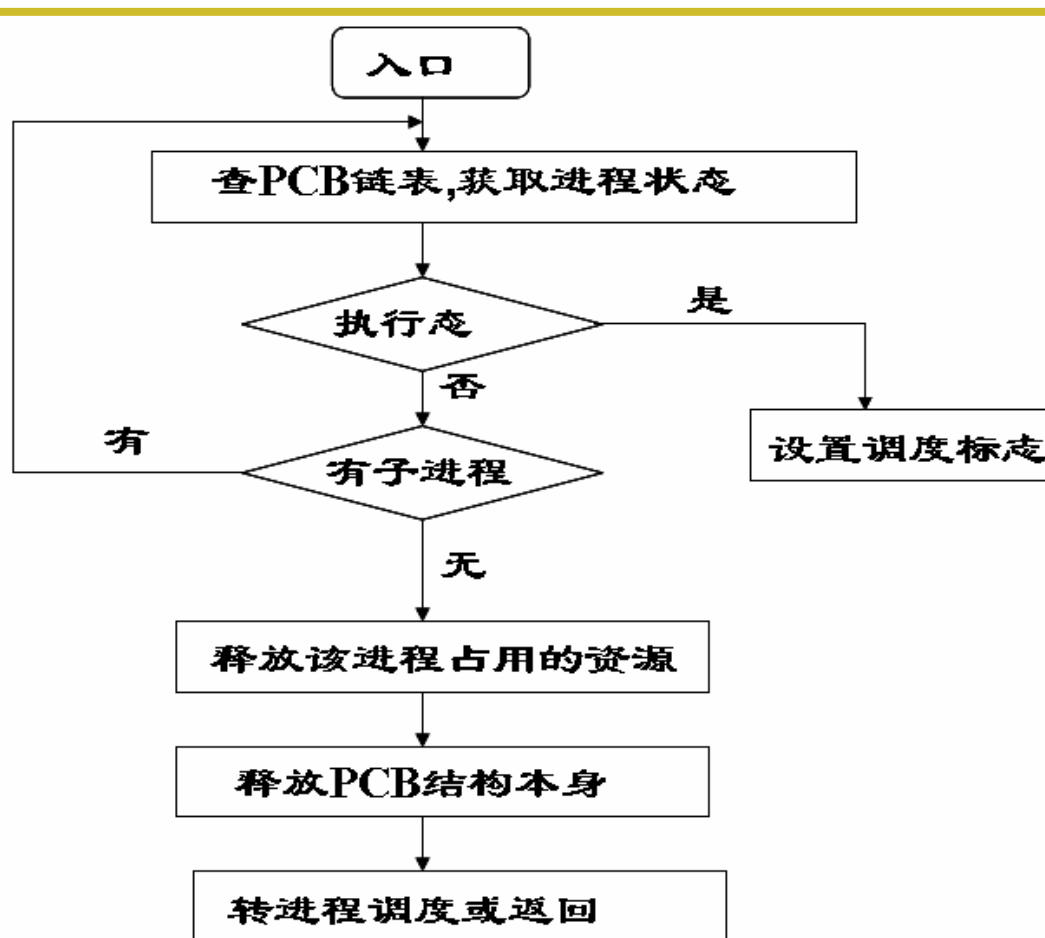
异常终止:进程运行中发生某种异常事件，在该事件中断处理过程中终止该进程

外界干预:外界强行干预要求终止该进程，诸如OS或操作员干预，父进程有权终止子进程



§ 4 进程控制(交通切换)

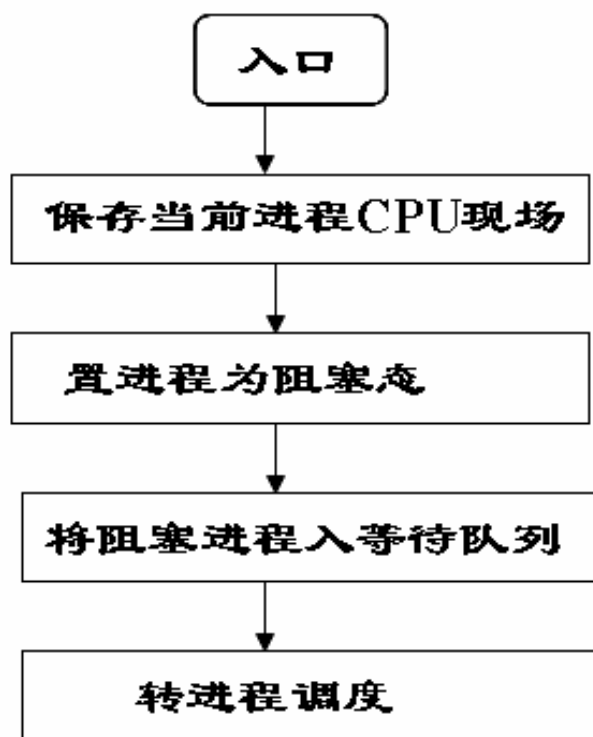
四、撤销原语



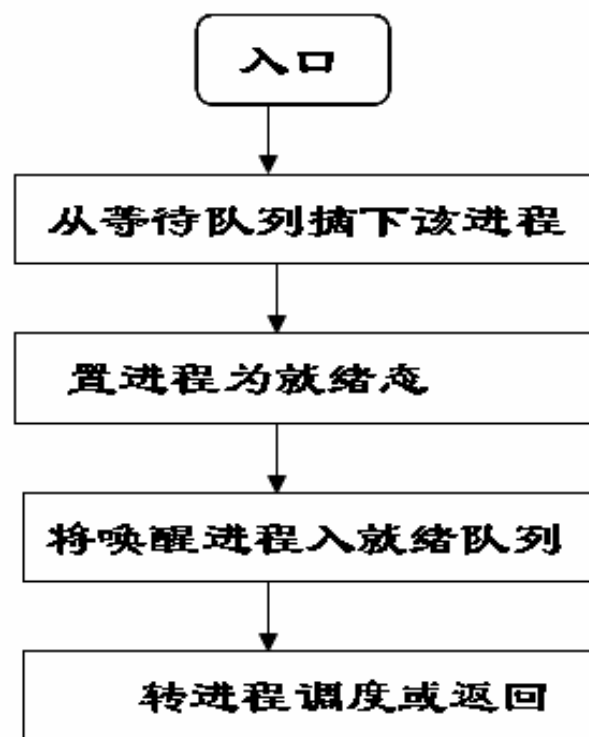


§ 4 进程控制(交通切换)

五、阻塞和唤醒原语



阻塞原语流程



唤醒原语流程



§ 4 进程控制(交通切换)

例1：以下 **D** 不属于进程控制的功能。

- A. 创建并发进程
- B. 夭折一个出错进程
- C. 唤醒一个阻塞进程
- D. 为进程分配CPU

例2：下面所述步骤中， **A** 不是创建进程所必需的。

- A. 由调度程序为进程分配CPU
- B. 建立一个进程控制块
- C. 为进程分配内存
- D. 将进程控制块链入就绪队列

例3：判断题：

(1)原语可以被多个进程同时执行。 (**F**)

(2)父进程终止,子进程可以不必随之撤消。 (**F**)

例4：下列选项中，导致创建新进程的操作是 **C**。

- I. 用户登陆成功
 - II. 设备分配
 - III. 启动程序执行
- A. 仅I和II B. 仅II和III C. 仅I和III D. I、II、III



§ 4 进程控制(交通切换)


例5 (北方交通大学)：进程控制的功能是首先为将要参加并发执行的程序 A，进程完成时撤消该进程，以及控制进程的 D。进程控制通常是利用 F 实现的，进程从运行态到阻塞态的转换，由 I 的进程调用 G 原语来实现；一个进程因等待某类资源而阻塞，正在执行的进程释放该类资源时调用 H 原语把阻塞的进程转换为 K。正在执行的进程响应外中断后再把阻塞的进程唤醒，被唤醒的进程原来等待的事件为 J。


- | | | | |
|---------|----------|---------|---------|
| A. 创建进程 | B. 分派CPU | C. 调入内存 | D. 状态转换 |
| E. 过程调用 | F. 原语 | G. 阻塞 | H. 唤醒 |
| I. 正在运行 | J. I/O操作 | K. 就绪态 | L. 运行态 |




§ 4 进程控制(交通切换)


例5：在进程控制中，导致进程调度的时机有哪些？

 **现行进程正常结束或异常终止**

 **现行进程因请求I/O等原因自行阻塞**

 **时间片到，现行进程CPU被剥夺**

 **可抢占调度中，更紧迫进程到达就绪队列**

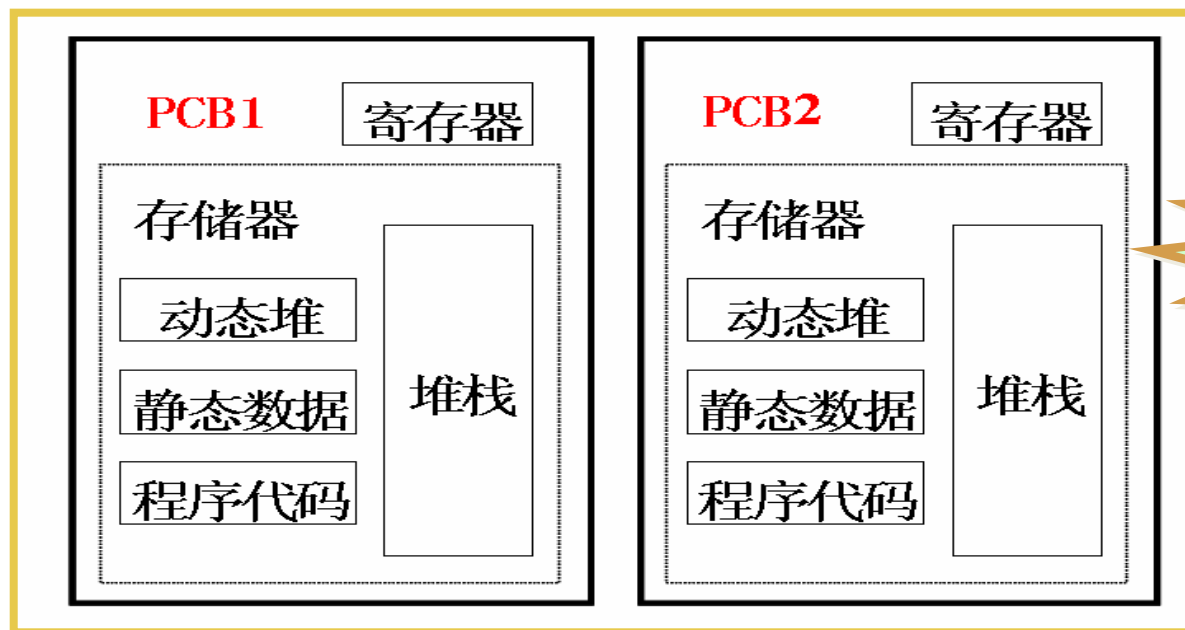
 **挂起了现行进程或激活了一个静止就绪进程**



§ 5 线程机制

一、引入线程的动机

- **进程的独立性**：独立的资源分配单位和CPU调度单位。
- **线程**：是进程内派生出的一个独立的运行线索,同一进程中的线程共享所隶属进程的主存及其它资源。进程仍是资源分配单位。
 - 只拥有必不可少的资源：TCB、少量寄存器上下文和栈
 - 同样具有就绪、阻塞和执行三种基本状态



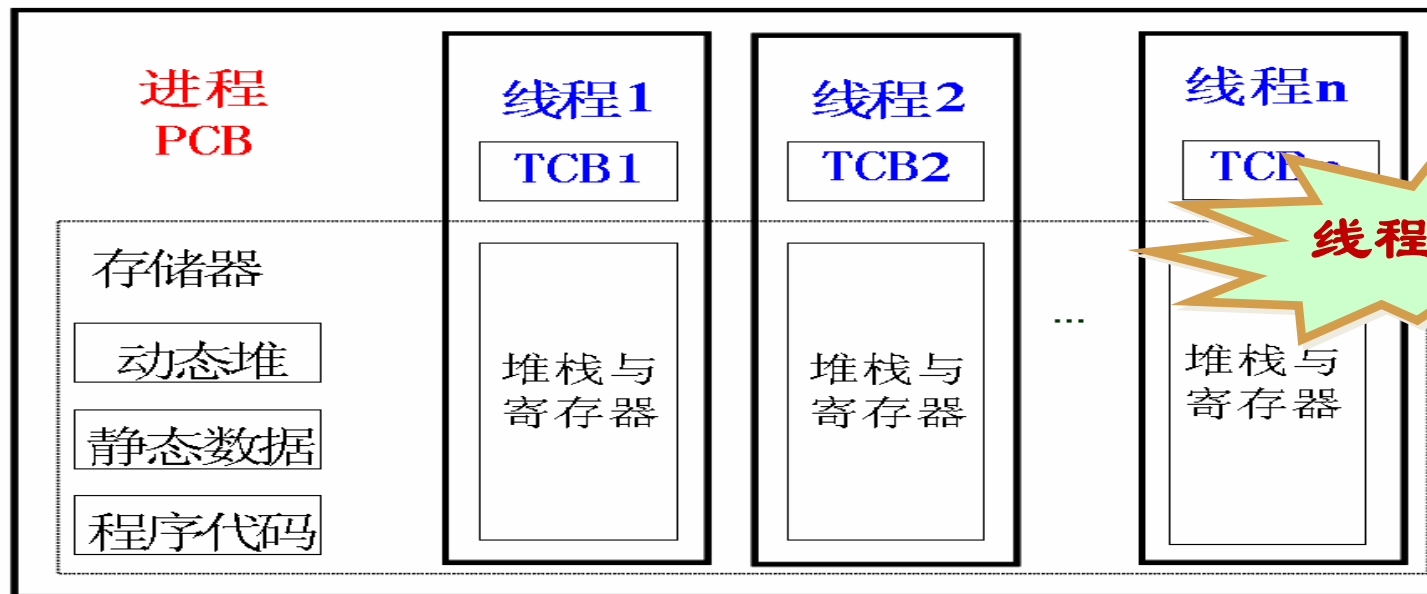
进程间关系



§ 5 线程机制

一、引入线程的动机

- **进程的独立性**：独立的资源分配单位和CPU调度单位。
- **线程**：是进程内派生出的一个独立的运行线索,同一进程中的线程共享所隶属进程的主存及其它资源。进程仍是资源分配单位。
 - 只拥有必不可少的资源：TCB、少量寄存器上下文和栈
 - 同样具有就绪、阻塞和执行三种基本状态





§ 5 线程机制

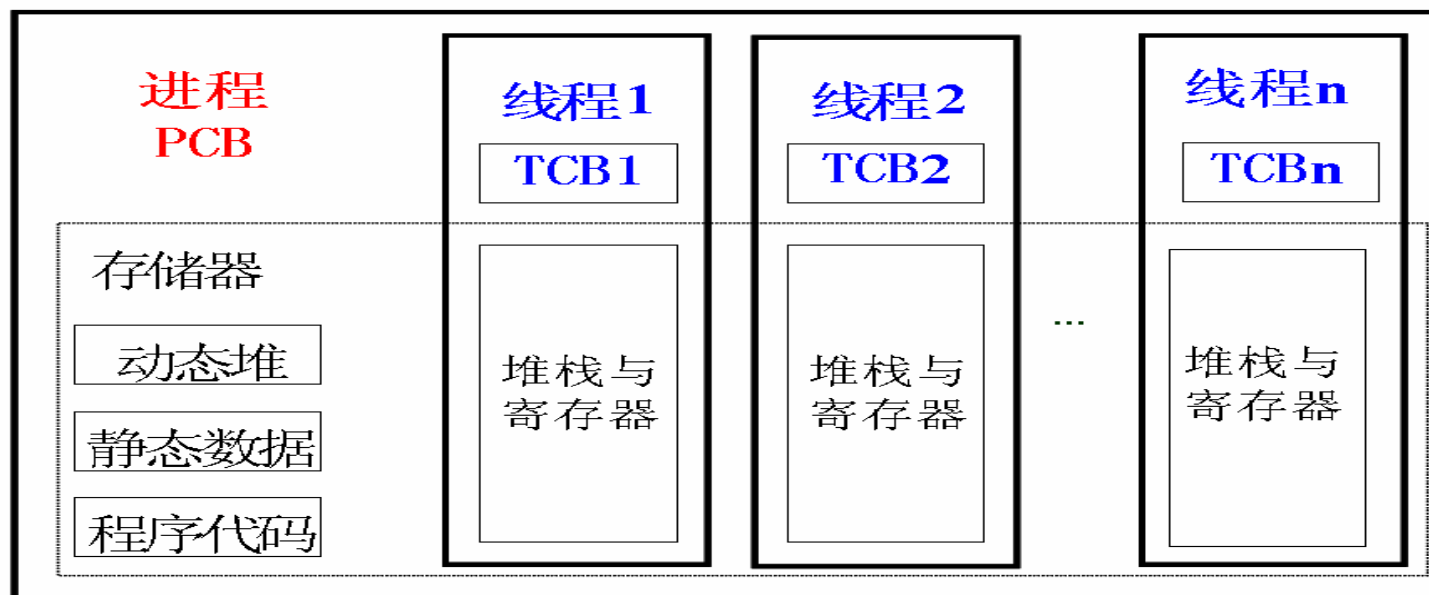
一、引入线程的动机

● **线程的优点：降低了管理并发的时空开销，简化了线程间通信，有利于提高并发度、获得更好的并发性**

——线程的创建时间和终止时间比进程短

——同一进程内的线程切换时间比进程短

——同一进程内的线程间可直接进行不通过内核的通信





§ 5 线程机制

二、进程和线程的区别与联系

1.从调度上

线程切换需要切换的内容远少于进程，线程切换速度快于进程

2.拥有资源

进程是资源的独立拥有者，进程管理的时空开销大；同一进程中的线程共享所属进程的资源，故线程管理开销小。

3.从通信上

同一进程中的线程共享主存和文件资源，其通信不需要借助内核功能。通信效率高、成本低。

4.并发性（粒度）和并发度

线程的引入使并发渗透到进程内，系统并发性更好。
小的系统开销使系统的并发度更高(eg:UNIX OS 40~100个Process,OS/2支持4096个Thread)。



§ 5 线程机制

三、线程实现机制

1.用户级线程ULT(User-Level Thread)

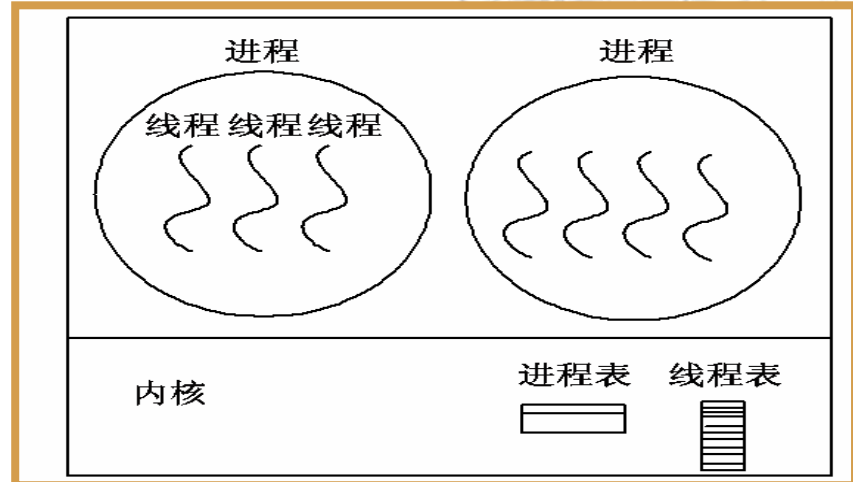
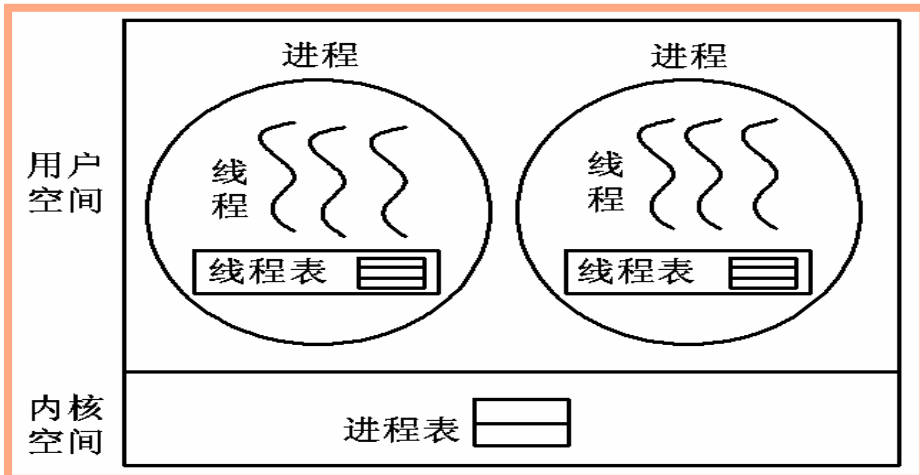
线程的创建、撤消以及切换不依赖内核,由用户空间的线程库管理线程。核心只感知进程,不了解线程的存在。eg.UNIX等

2.核心级线程KLT/KST(Kernel-Level/Supported Thread)

线程的创建、撤消以及切换均由OS内核完成。核心既能感知进程,也能感知线程的存在。eg.Mach、Windows NT等

3.组合方式

同时支持ULT和KLT方式。核心可感知核心级线程。eg.Linux、solaris等。





§ 5 线程机制

四、ULT & KLT比较

•线程切换

ULT: 线程切换不调用核心, 切换速度快(**优**)

KLT: 线程间的切换需陷入内核, 导致速度下降(**缺**)

•系统调用(往往是阻塞的)

ULT: 当线程调用系统调用时, 整个进程阻塞(**缺**)

KLT: 进程中一个线程被阻塞, 不影响其它线程的运行(**优**)

•线程调度算法

ULT: 调度是应用程序特定的, 线程调度算法可针对应用优化

KLT: 时间片分配给线程, 多线程的进程可获得更多CPU时间



§ 5 线程机制

例1：采用纯用户级多线程策略时，处理器的调度对象是 A；采用混合式多线程策略时，处理器调度的对象是 D。

A. 进程

B. 作业

C. 用户级线程

D. 内核级线程

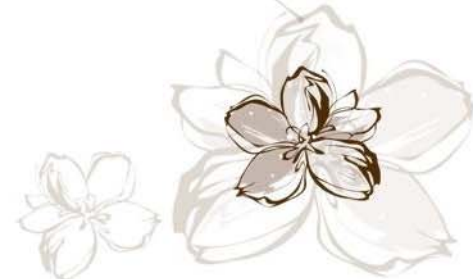
例2：在引入线程的操作系统中，资源分配的基本单位是 C，CPU分配的基本单位是 D。

A. 程序

B. 作业

C. 进程

D. 线程





§ 5 线程机制

例3 :在支持多线程的系统中，进程P创建的若干线程不能共享的是 D。

- A. 进程的代码段
- B. 进程P中打开的文件
- C. 进程P的全局变量
- D. 进程P中某线程的栈指针

例4 :下列关于进程和线程的叙述中，正确的是 A。

- A. 不管系统是否支持线程，进程都是资源分配的基本单位
- B. 线程是资源分配的基本单位，进程是调度的基本单位
- C. 系统级线程和用户级线程的切换都需要内核的支持
- D. 同一进程中的各个线程拥有各自不同的地址空间





§ 5 线程机制

例5: 分时系统中，条件相同的情况下，通常KLT会比ULT得到更多的CPU时间，请简要解释。

以进程A包含一个线程，进程B包含10个线程为例。

ULT系统中的调度以进程为单位，则进程A、B得到一样的时间片。多线程的进程中每个线程推进就慢。

若系统设置KLT，则其调度以线程为单位，则进程B获得的CPU时间是进程A的10倍。

这样，在条件相同的情况下，多线程的进程在KLT方式下比ULT得到更多的CPU时间。

