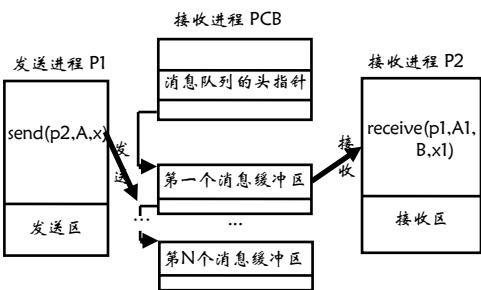


### 3.3 消息传递通信(1)



61

### 3.3 消息传递通信(2)

- 消息队列通常放在进程控制块中

- PCB的结构

```
struct PCB {  
    ...  
    mq;      //消息队列队首指针  
    mutex;   //消息队列互斥信号量  
    sm;      //消息队列同步信号量  
    ...  
}
```

62

### 3.3 消息传递通信(3)

#### ➢ 消息传递通信

- 消息结构
- 消息缓冲
- 直接/间接通信
- 同步/异步通信

#### ➢ 直接通信

- 消息发送原语
  - send(接收者, 被发送信息始址, 信息长度)

```
struct message_buffer {  
    sender:消息发送者  
    receiver:消息接收者  
    size:消息长度  
    text:消息正文  
    next:指向下一消息  
    缓冲区指针  
};
```

63

### 3.3 消息传递通信(4)

#### • 主要工作

- 请求分配一个消息缓冲区；
- 将消息正文传递到该缓冲区中，并填入有关发送参数；
- 将该消息缓冲区挂到接收进程消息链上

```
send(receiver,a) {  
    getbuf(a.size,i);  
    i.sender=a.sender;  
    i.size=a.size;  
    i.text=a.text;  
    i.next=0;  
    getid(PCBset,receiver,j);  
    P(j.mutex);  
    Insert(j.mq,i);  
    V(j.mutex);  
    V(j.sm);  
}
```

64

### 3.3 消息传递通信(5)

#### • 接收消息原语

- receive (发送者, 信息始址, 接收区始址, 信息长度)

#### • 主要工作

- 检查消息链上是否有消息；
- 有则将消息接收到接收区；
- 无则阻塞等待消息的到来

```
receive(b) {  
    j=getid;  
    P(j.sm);  
    P(j.mutex);  
    Remove(j.mq,i);  
    V(j.mutex);  
    b.sender=i.sender;  
    b.size=i.size;  
    b.text=i.text;  
}
```

65

### 3.3 消息传递通信(6)

#### ➢ 间接通信 (信箱/端口通信)

#### • 间接通信方式

- 发送原语
  - send(A, message)
- 接收原语
  - receive(A, message)
- 信箱属性
  - 共享
  - 专用

66

### 3.3 消息传递通信(7)

#### ➤ 消息同步

- 发送进程与接收进程状态
  - 阻塞
  - 非阻塞
- 消息同步方式
  - 非阻塞发送, 非阻塞接收
  - 非阻塞发送, 阻塞接收
  - 阻塞发送, 阻塞接收
  - 阻塞发送, 非阻塞接收

67

### 3.4 客户-服务器通信

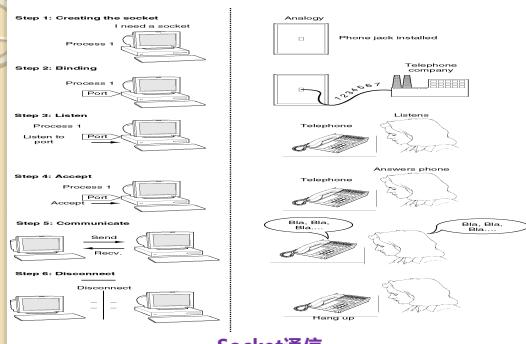
#### ➤ 3.4.1 Socket

#### ➤ 3.4.2 RPC(Remote Procedure Call)

#### ➤ 3.4.3 Pipe

68

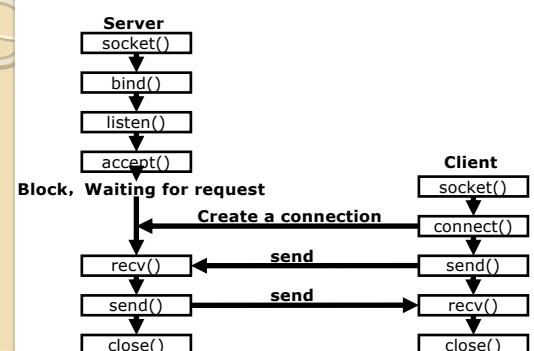
#### 3.4.1 Socket (1)



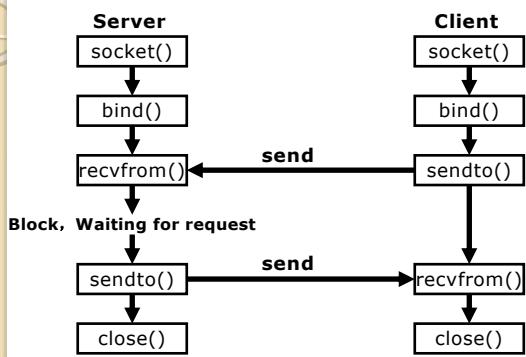
Socket通信

69

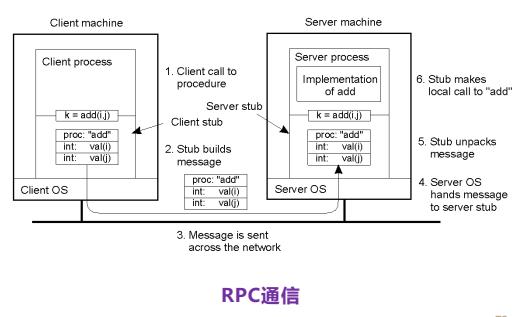
#### 3.4.1 Socket (2)



#### 3.4.1 Socket (3)



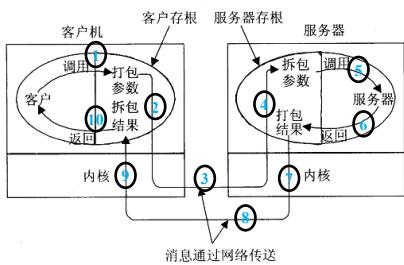
#### 3.4.2 RPC(1)



72

### 3.4.2 RPC(2)

#### ➤ RPC的步骤



### 3.4.2 远程过程调用(3)

#### ➤ 主要问题

- 参数传递
- 参数表示
- 客户-服务器绑定

### 3.4.3 Pipe(1)

#### ➤ 管道(Pipe)

- 概念
  - 连接一个读进程和一个写进程，允许它们以生产者-消费者方式进行通信的一个文件（文件大小受限）
- 几个问题
  - 类型：有名管道 VS 无名管道
  - 通信方式：单向 VS 双向（半双工 VS 全双工）
  - 通信进程间是否存在父子关系
  - 是否可以在网络上使用管道

75

### 3.4.3 Pipe(2)

- 无名管道（普通管道）
  - 创建
    - 由进程创建的临时文件
    - 无法从创建它的进程外部访问
    - 父进程创建一个管道，并使用该管道与其创建的子进程进行通信
  - 通信
    - 以标准生产者-消费者方式进行通信
    - 单向通信，互斥访问管道
      - 生产者写入一端（管道写入端）
      - 消费者从另一端读取（管道读取端）

76

### 3.4.3 Pipe(3)

- 有名管道
  - 创建
    - 由系统创建的长期存在的文件
    - 所有进程可通过路径名访问该管道
    - 所有进程均可访问有名管道
  - 通信
    - 双向通信
    - 半双工

77

### 3.5 死锁

- 3.5.1 基本概念
- 3.5.2 鸽鸟算法
- 3.5.3 死锁预防
- 3.5.4 死锁避免
- 3.5.5 死锁检测
- 3.5.6 死锁恢复

78

### 3.5.1 基本概念(1)

#### ➤ 死锁概念

- 多个进程在运行过程中因争夺资源而造成的一种僵局，当进程处于这种僵持状态时，若无外力作用，所有进程都将无法向前推进

#### ➤ 产生死锁的原因

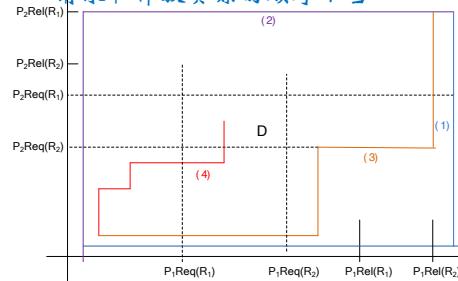
- 竞争资源（资源数量不足）
- 资源类型
  - 可抢占资源：主存、CPU
  - 不可抢占资源：慢速设备/共享变量

79

### 3.5.1 基本概念(2)

#### • 进程推进顺序非法

##### • 请求和释放资源的顺序不当



80

### 3.5.1 基本概念(3)

#### ➤ 死锁产生的必要条件

##### 同时具备下列四个条件

- 互斥条件
  - 每个资源是不可共享的，它或者已经分配给一个进程，或者空闲
- 不可剥夺条件
  - 进程所获得的资源在未使用完之前，不能被其它进程强行剥夺，只能由获得资源的进程自己释放
- 保持和等待条件
- 循环等待条件

81

### 3.5.1 基本概念(4)

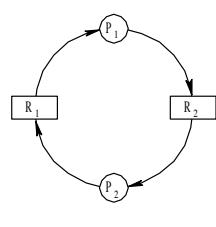
#### ➤ 死锁产生的必要条件

##### 同时具备下列四个条件

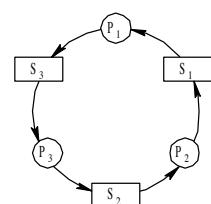
- 互斥条件
- 不可剥夺条件
- 保持和等待条件
  - 进程因请求资源而被阻塞等待时，对已经分配给它的资源保持不放
- 循环等待条件
  - 存在进程循环链，链中有两(多)个进程在等待链中下一个成员保持的资源

82

### 3.5.1 基本概念(5)



I/O设备共享时的死锁



进程之间通信时的死锁

83

### 3.5.1 基本概念(6)

#### ➤ 资源分配图

- 由节点V和边E构成；
- 节点V分为两类：
  - 进程节点： $P = \{P_1, P_2, \dots, P_n\}$ ，用圆圈表示，构成了系统中的进程集合；
  - 资源节点： $R = \{R_1, R_2, \dots, R_m\}$ ，用方块表示，表示系统中的各种资源；
- 边E分为两类：
  - 请求边 - 有向边  $P_i \rightarrow R_j$
  - 分配边 - 有向边  $R_j \rightarrow P_i$

84

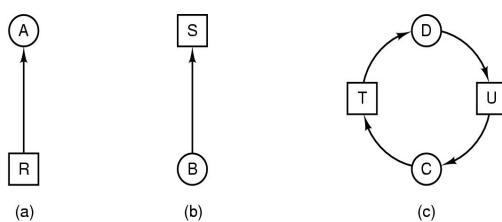
### 3.5.1 基本概念(7)

- 示例
  - 进程
  - 具有4个实例的资源
  - $P_i$  请求资源  $R_j$  的一个实例
  - $P_i$  拥有资源  $R_j$  的一个实例



85

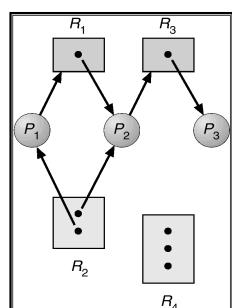
### 3.5.1 基本概念(8)



资源分配图示例

86

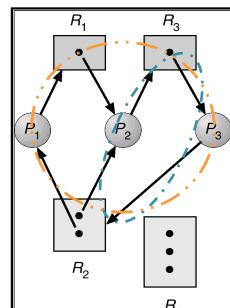
### 3.5.1 基本概念(9)



资源分配图示例

87

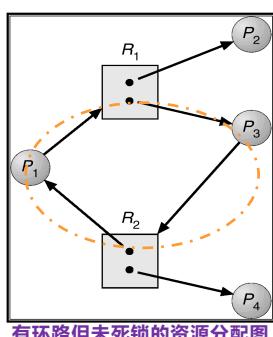
### 3.5.1 基本概念(10)



带有死锁的资源分配图

88

### 3.5.1 基本概念(11)



有环路但未死锁的资源分配图

89

### 3.5.1 基本概念(12)

- 结论
  - 资源分配图中无环路  $\Rightarrow$  无死锁
  - 资源分配图中有环路  $\Rightarrow$ 
    - 每类资源只有一个实例则死锁；
    - 每类资源有多个实例，可能死锁；
- 解决死锁的方法
  - 忽略死锁问题，假定系统永不死锁
  - 保证系统永远不进入死锁状态
    - 死锁预防
    - 死锁避免

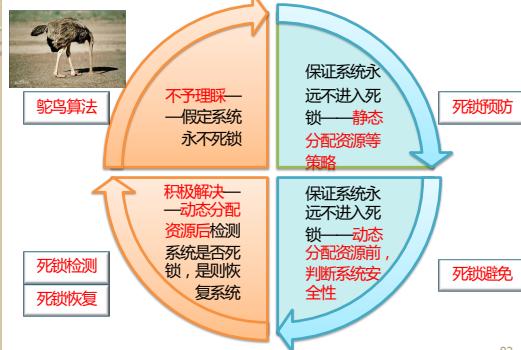
90

### 3.5.1 基本概念(13)

- 允许系统进入死锁状态，并可从死锁状态中恢复
  - 死锁检测
  - 死锁恢复

91

### 3.5.1 基本概念(14)



92

### 3.5.2 鸵鸟算法

- 假定死锁从不发生
- 原因
  - 死锁在计算机中很少出现
  - 预防死锁的代价太高

93

### 3.5.3 死锁预防(1)

- 破坏死锁产生的必要条件
- 破坏互斥条件
  - 资源特性
  - 将独享设备改造为共享设备
- 破坏非剥夺条件
  - 当一个已经占有了一些资源的进程提出新的资源申请而不能立即得到满足时，必须释放已经占有的全部资源，待以后需要时再重新申请

94

### 3.5.3 死锁预防(2)

- 缺点
  - 保护进程放弃资源时的现场及之后的恢复现场，系统要付出很高的代价
  - 增加了进程周转时间
- 破坏保持和请求条件
  - 进程在开始运行前必须获得所需的全部资源。若系统不能满足，则该进程等待
  - 优点
    - 简单，易于实现，安全

95

### 3.5.3 死锁预防(3)

- 缺点
  - 资源利用率低
  - 饥饿问题
- 破坏循环等待条件
  - 将系统全部资源按类进行全局编号排序，进程对资源的请求必须按照资源的递增或递减顺序序号申请
- 缺点
  - 找到能满足所有进程要求的资源编号
  - 限制新类型设备的增加

96

### 3.5.4 死锁避免(1)

#### ➤ 死锁预防

- 静态分配资源

#### ➤ 死锁避免

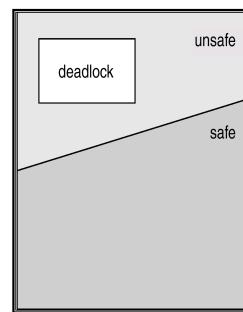
- 动态分配资源

- 允许进程动态地申请资源，一次申请一部分资源。系统在进行资源分配之前，先计算资源分配的安全性。若此次分配不会导致系统进入不安全状态，便将资源分配给进程；否则，进程等待

97

### 3.5.4 死锁避免(2)

- 系统处于安全状态  $\Rightarrow$  无死锁
- 系统处于不安全状态  $\Rightarrow$  可能死锁
- 避免死锁  $\Rightarrow$  保证系统永不进入不安全状态



98

### 3.5.4 死锁避免(3)

#### ➤ 安全状态

- 系统能按照某种进程顺序( $P_1, P_2, \dots, P_n$ )来为每个进程 $P_i$ 分配其所需资源，直至满足每个进程对资源的最大需求，使每个进程都可顺利地完成

#### ➤ 安全序列

- $(P_1, P_2, \dots, P_n)$

#### ➤ 不安全状态

- 不存在一个安全序列

99

### 3.5.4 死锁避免(4)

| Has Max |
|---------|---------|---------|---------|---------|
| A 3 9   | A 3 9   | A 3 9   | A 3 9   | A 3 9   |
| B 2 4   | B 4 4   | B 0 -   | B 0 -   | B 0 -   |
| C 2 7   | C 2 7   | C 2 7   | C 7 7   | C 0 -   |
| Free: 3 | Free: 1 | Free: 5 | Free: 0 | Free: 7 |

(a) (b) (c) (d) (e)

安全状态——安全序列(B,C,A)

100

### 3.5.4 死锁避免(5)

Has Max	Has Max	Has Max	Has Max
A 3 9	A 4 9	A 4 9	A 4 9
B 2 4	B 2 4	B 4 4	B - -
C 2 7	C 2 7	C 2 7	C 2 7

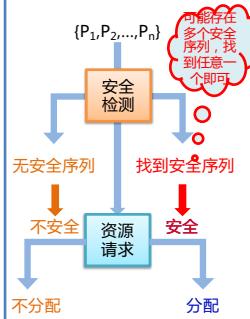
Free: 3 Free: 2 Free: 0 Free: 4  
(a) (b) (c) (d)

不安全状态

101

### 3.5.4 死锁避免(6)

- ✓ 银行家算法
- ✓ 若系统中有n个进程( $P_1, P_2, \dots, P_n$ )，当某一进程提出资源申请时，若系统能够找到一个安全状态，则满足该进程的资源请求，否则该进程等待
- ✓ 安全状态：若系统能够按照某种顺序( $P_1, P_2, \dots, P_n$ )为每个进程 $P_i$ ( $i=1..n$ )分配其所需资源，直至满足每个进程对资源的最大需求，使每个进程都可顺利地完成，则系统处于安全状态
  - ✓  $(P_1, P_2, \dots, P_n)$ 是一个安全序列
  - ✓ 不安全状态：不存在安全序列



102

### 3.5.4 死锁避免(7)

➤ 银行家算法

- 1965年由Dijkstra提出
- 数据结构
  - $n$ 表示进程数目,  $m$ 表示资源类型
  - 可用资源向量Available: 含有 $m$ 个元素的数组, 每个元素代表一类可利用的资源的数目, 初始值为系统中所配置的该类资源的全部可用数目。如果Available[j]=k, 则表示系统中现有 $R_j$ 类资源 $k$ 个。

103

### 3.5.4 死锁避免(8)

- 最大需求矩阵Max:  $n \times m$ 的矩阵, 定义了系统中 $n$ 个进程中的每一个进程对 $m$ 类资源的最大需求。如果Max[i,j]=k, 则表示进程 $P_i$ 需要 $R_j$ 类资源的最大数目为 $k$ 。
- 分配矩阵Allocation:  $n \times m$ 的矩阵, 定义了系统中每一类资源当前已分配给每一进程的资源数目。如果Allocation[i,j]=k, 则表示进程 $P_i$ 当前已分配到 $R_j$ 类资源的数目为 $k$ 。

104

### 3.5.4 死锁避免(9)

- 需求矩阵Need:  $n \times m$ 的矩阵, 表示每一个进程还需要的各类资源的数目。如果Need[i,j]=k, 则表示进程 $P_i$ 还需要 $R_j$ 类资源 $k$ 个, 才能完成其任务。
- 各个矩阵之间的关系:  
 $Need[i,j] = Max[i,j] - Allocation[i,j]$
- 算法
  - 设Request<sub>i</sub>是进程 $P_i$ 的请求向量, 如果Request<sub>i</sub>[j]=k, 表示进程 $P_i$ 需要 $k$ 个 $R_j$ 类的资源

105

### 3.5.4 死锁避免(10)

1. 如果Request<sub>i</sub>[j] ≤ Need[i,j], 转向第2步; 否则出错;
2. 如果Request<sub>i</sub>[j] ≤ Available[j], 转向第3步; 否则表示尚无足够资源,  $P_i$ 须等待;
3. 假定将资源分配给进程 $P_i$ , 修改下列数据结构:  
 $Available[j] = Available[j] - Request_i[j]$   
 $Allocation[i,j] = Allocation[i,j] + Request_i[j]$   
 $Need[i,j] = Need[i,j] - Request_i[j]$

106

### 3.5.4 死锁避免(11)

4. 系统执行安全性算法, 检查此次分配后, 系统是否处于安全状态。
  - 安全 ⇒ 资源分配给进程 $P_i$
  - 不安全 ⇒  $P_i$ 等待, 本次试探分配取消, 恢复原来的资源分配状态
- 安全性算法
  1. 设置工作向量Work, 表示系统可提供给进程继续运行所需要的各类资源数目, 它含有 $m$ 个元素, 初始值为Work=Available;

107

### 3.5.4 死锁避免(12)

2. 设置向量Finish, 表示系统是否有足够的资源分配给进程使之运行完成, 它含有 $n$ 个元素, 初始值为Finish[i]=false;
  3. 从进程集合中查找能满足下列条件的进程
    - Finish[i]=false
    - Need[i,j] ≤ Work[j]
- 若存在该进程, 则转4; 否则, 执行5;

108

### 3.5.4 死锁避免(13)

4. 进程 $P_i$ 获得资源后可顺利执行至完成，并释放分配给它的资源，即执行

$$\text{Work}[j] = \text{Work}[j] + \text{Allocation}[i,j]$$

$$\text{Finish}[i] = \text{true}$$

执行第3步：

5. 若所有进程均满足 $\text{Finish}[i] == \text{true}$ ，则系统处于安全状态；否则为不安全状态

➤ 示例：假定有5个进程 $\{P_0, P_1, P_2, P_3, P_4\}$ ，3类资源 $\{A, B, C\}$ ，各类资源的数目分别为10, 5, 7。

109

### 3.5.4 死锁避免(14)

➤ T0时刻的资源分配情况如下：

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
$P_0$	7	5	3	0	1	0	7	4	3	3	3	2
$P_1$	3	2	2	2	0	0	1	2	2			
$P_2$	9	0	2	3	0	2	6	0	0			
$P_3$	2	2	2	2	1	1	0	1	1			
$P_4$	4	3	3	0	0	2	4	3	1			

110

### 3.5.4 死锁避免(15)

➤ 存在安全序列 $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ ，系统处于安全状态

	Work			Allocation			Need			Work+Allocation			Finish
	A	B	C	A	B	C	A	B	C	A	B	C	
$P_1$	3	3	2	2	0	0	1	2	2	5	3	2	true
$P_3$	5	3	2	2	1	1	0	1	1	7	4	3	true
$P_4$	7	4	3	0	0	2	4	3	1	7	4	5	true
$P_2$	7	4	5	3	0	2	6	0	0	10	4	7	true
$P_0$	10	4	7	0	1	0	7	4	3	10	5	7	true

111

### 3.5.4 死锁避免(16)

➤  $P_1$ 请求资源 $(1,0,2)$

- $\text{Request}_1(1,0,2) \leq \text{Need}_1(1,2,2) \Rightarrow \text{true}$
- $\text{Request}_1(1,0,2) \leq \text{Available}_1(3,3,2) \Rightarrow \text{true}$

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
$P_0$	7	5	3	0	1	0	7	4	3	2	3	0
$P_1$	3	2	2	3	0	2	0	2	0			
$P_2$	9	0	2	3	0	2	6	0	0			
$P_3$	2	2	2	2	1	1	0	1	1			
$P_4$	4	3	3	0	0	2	4	3	1			

112

### 3.5.4 死锁避免(17)

➤ 执行安全性算法可知，存在安全序列 $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ ，系统处于安全状态，可将资源分配给 $P_1$

	Work			Allocation			Need			Work+Allocation			Finish
	A	B	C	A	B	C	A	B	C	A	B	C	
$P_1$	2	3	0	3	0	2	0	2	0	5	3	2	true
$P_3$	5	3	2	2	1	1	0	1	1	7	4	3	true
$P_4$	7	4	3	0	0	2	4	3	1	7	4	5	true
$P_0$	7	4	5	0	1	0	7	4	3	7	5	5	true
$P_2$	7	5	5	3	0	2	6	0	0	10	5	7	true

113

### 3.5.4 死锁避免(18)

➤  $P_4$ 请求资源 $(3,3,0)$

- $\text{Request}_4(3,3,0) \leq \text{Need}_4(4,3,1) \Rightarrow \text{true}$
- $\text{Request}_4(3,3,0) \leq \text{Available}_4(2,3,0) \Rightarrow \text{false}$
- $P_4$ 等待

➤  $P_0$ 请求资源 $(0,2,0)$

- $\text{Request}_0(0,2,0) \leq \text{Need}_0(7,4,3) \Rightarrow \text{true}$
- $\text{Request}_0(0,2,0) \leq \text{Available}_0(2,3,0) \Rightarrow \text{true}$
- 执行安全性算法可知，系统进入不安全状态，不能够将资源分配给 $P_0$
- $P_0$ 等待

114

### 3.5.4 死锁避免(19)

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
$P_0$	7	5	3	0	3	0	7	2	3	2	1	0
$P_1$	3	2	2	3	0	2	0	2	0			
$P_2$	9	0	2	3	0	2	6	0	0			
$P_3$	2	2	2	2	1	1	0	1	1			
$P_4$	4	3	3	0	0	2	4	3	1			

115

### 3.5.5 死锁检测(1)

➤ 系统在进行资源分配之后，计算资源分配的安全性。若此次分配会导致系统死锁，则采取措施恢复；否则，继续。

➤ 对系统中的进程资源分配图进行检测，若图中包含了一个或多个循环，则存在死锁，否则不存在死锁。

#### ➤ 检测时机

- 每次资源请求时检测

- + 较早检测到

- + 浪费大量CPU时间

- 间隔一段时间检测

116

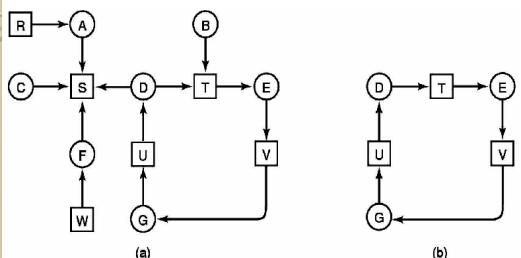
### 3.5.5 死锁检测(2)

#### ➤ 示例：

- 进程A保持资源R，请求资源S；
- 进程B没有保持资源，正在请求资源T；
- 进程C没有保持资源，正在请求资源S；
- 进程D保持资源U，正请求资源S和T；
- 进程E保持资源T，请求资源V；
- 进程F保持资源W，请求资源S；
- 进程G保持资源V，请求资源U；

117

### 3.5.5 死锁检测(3)



带有死锁的进程资源图

118

### 3.5.5 死锁检测(4)

#### ➤ 检测算法（每类资源一个单位数量）

1. 以图中的每个节点N为起始节点，分别执行下列步骤：
2. 将L初始化为空，表示所有弧均未标记。
3. 将当前节点加到L的末端，检查该节点在L中是否出现过。如果出现，这个图包含一个环路，算法终止。如果没有，转4。
4. 检查该节点是否有未标记的引出弧。如果有，转5；若没有，转6。

119

### 3.5.5 死锁检测(5)

5. 任意选择一个未标记的引出弧对其进行标记，标记后将引出弧所到节点作为新的当前节点，转3。

6. 若所有从该节点引出的弧都已标记，则返回至前一节点；如果该节点是最初开始的节点，则该图没有包含环路，算法终止；若该节点不是初始节点，以该节点作为当前节点，转4。

#### ➤ 示例：R节点

$$\cdot L = []$$

$$\cdot L = [R, A]$$

$$L = [R]$$

$$L = [R, A, S]$$

120

### 3.5.5 死锁检测(6)

- 示例：B节点
  - L = []
  - L = [B]
  - L = [B, T]
  - L = [B, T, E]
  - L = [B, T, E, V]
  - L = [B, T, E, V, G]
  - L = [B, T, E, V, G, U]
  - L = [B, T, E, V, G, U, D]
  - L = [B, T, E, V, G, U, D, T]

[21]

### 3.5.5 死锁检测(7)

- 检测算法（每类资源有多个）
  - 简化资源分配图
- 检查每个连接进程和资源的箭头：
  1. 如果一个资源只有射出箭头（无未满足的资源请求），则擦除所有与其相关的箭头。
  2. 如果某个进程只有指向它的箭头（所有资源请求均已满足），则擦除所有与其相关的箭头。

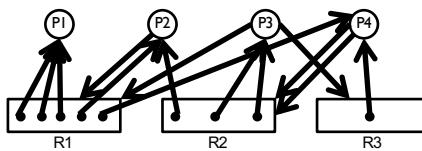
[22]

### 3.5.5 死锁检测(8)

- 3. 如果某个进程有射出的箭头，并且对于每个请求箭头都有一个可用的资源（无射出箭头），则擦除所有与其相关的箭头。
- 检测所有进程，如果至少有一个进程的箭头可以擦除，则返回重复上述处理过程，直到没有箭头剩余或没有进程的箭头可以擦除为止。
- 当且仅当没有箭头剩余，系统处于非死锁状态。

[23]

### 3.5.5 死锁检测(9)



[24]

### 3.5.6 死锁恢复

- 通过剥夺资源使系统恢复
  - 通过剥夺一些进程的资源使系统恢复
    - 最小代价
  - 通过滚回一些进程恢复系统
    - 将一个或多个死锁进程滚回
  - 饥饿问题
- 通过撤销进程使系统恢复
  - 撤销死锁进程，断开环路
  - 撤销不在环路中的非死锁进程

[25]

### 本章要点

- 进程通信方式
- 临界资源与临界区
- 信号量的意义及P/V操作
- 经典同步问题
- 管程
- 死锁的概念及产生死锁的必要条件
- 死锁处理策略
- 系统安全状态
- 银行家算法

[26]