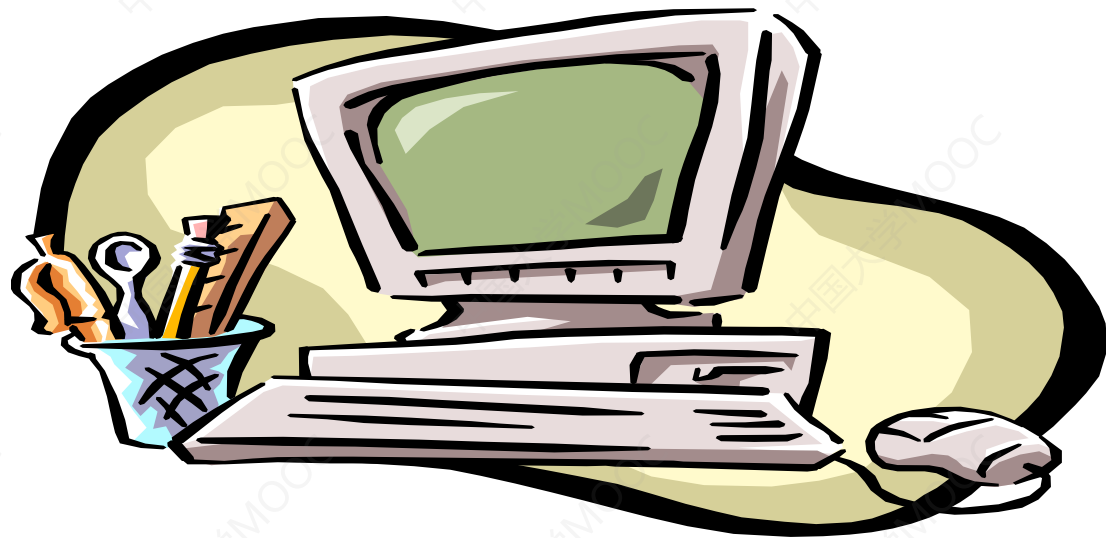


《操作系统》



主讲教师：翟高寿

联系电话：010-51684177 (办)

电子邮件：gszhai@bjtu.edu.cn

制作人：翟高寿

制作单位：北京交通大学计算机学院

第三章 处理机调度与死锁

3.1 高级、中级与低级调度

3.2 调度队列模型

3.3 调度方式与算法选择准则

3.4 调度算法

3.5 死锁产生及处理策略

3.6 死锁避免与银行家算法

多道程序环境与处理机调度

□ 作业类型与处理机获得过程

作业?

- 批量型作业、终端型作业。

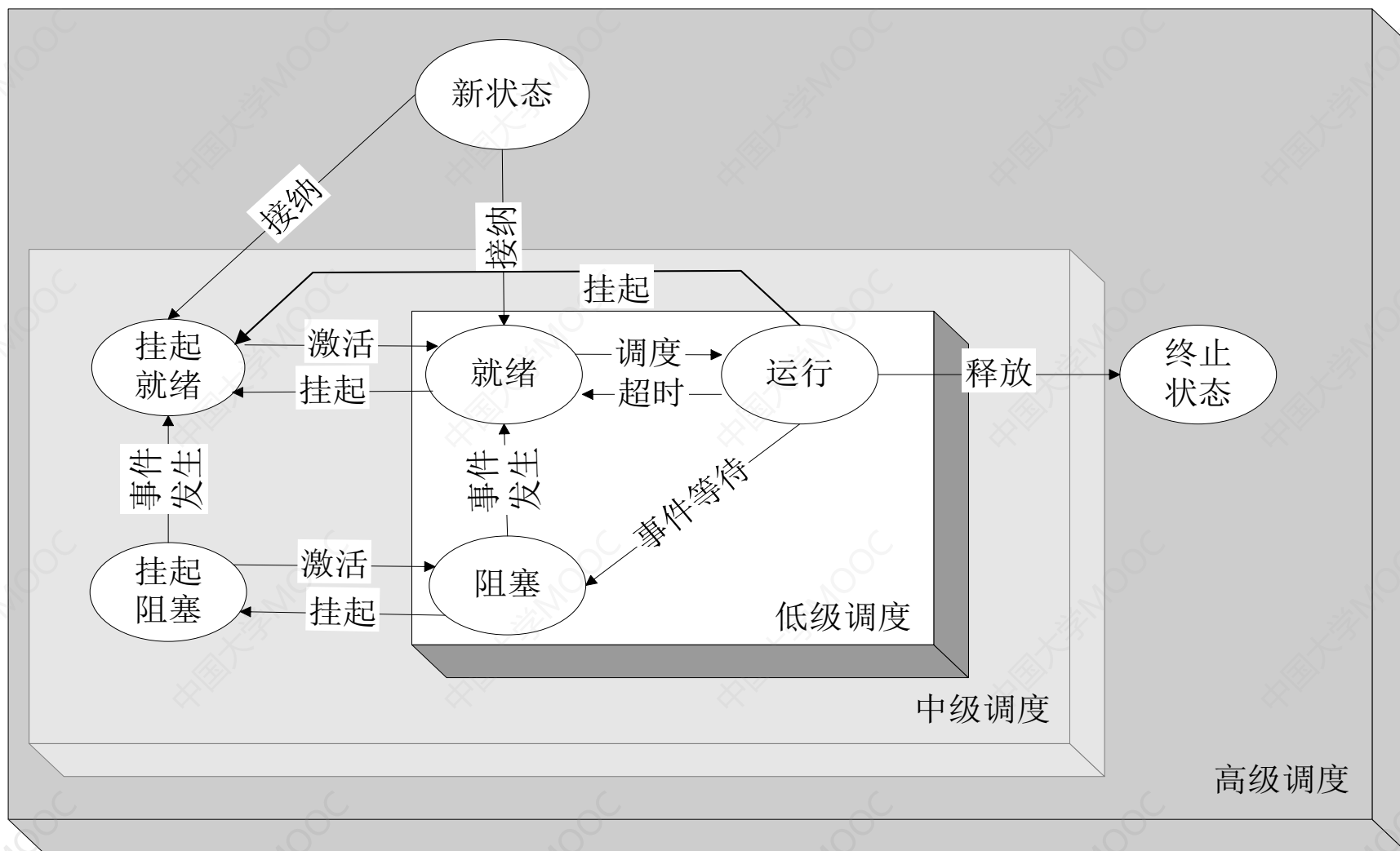
□ 基于操作系统类型的调度分类

- 批处理/分时/实时调度及多处理机调度

□ 调度是多道程序系统的关键所在

- 系统运行性能（如吞吐量大小、周转时间长短、响应及时性等）在很大程度上都取决于调度，特别是处理机调度
- 一个作业从提交到执行，通常都要经历高级、中级、低级及I/O等多级调度

多级调度示意图



高级调度（作业/长程/宏观调度）

□ 概念

- 用于决定把外存上处于后备队列中的哪些作业调入内存，并为它们创建进程和分配必要资源；然后，再将新创建进程插入到就绪队列上准备执行

□ 操作系统配置作业调度机制分析

- √ 批处理系统
- × 分时系统、实时系统及时性要求

□ 作业调度机制要领

- 作业量确定 ← 多道程序度(Degree of Multiprogramming)
- 作业选择 ← 调度算法

低级调度（进程/短程调度）

□ 概念

- 用来决定就绪队列中的哪个进程将获得处理机，然后再由分派程序（Dispatcher）执行把处理机分配给该进程的具体操作

□ 操作系统配置进程调度机制分析

- 基本调度，所有类型操作系统均需配置

□ 调度方式分类

- 非抢占方式（仅适用于批处理系统）
- 抢占方式（分时、实时及批处理系统均可）

非抢占与抢占调度方式比较

□ 非抢占调度方式(Non-preemptive Mode)

- 处理机分配给进程直至完成或阻塞
- 引起进程调度的因素：当前进程执行完毕或因发生事件、提出I/O请求、执行原语操作而阻塞
- 实现简单、系统开销小，但难以满足紧急任务要求，故不宜在实时系统中采用

□ 抢占调度方式(Preemptive Mode)

- 允许暂停正在执行进程和重新分配处理机
- 抢占原则（优先权/短作业优先/时间片原则）

中级调度（中程调度）

□ 概念

- 为提高内存利用率和系统吞吐量，应使那些暂时不能运行的进程放弃占用内存资源，即调至外存上去等待；当内存稍有空闲时，可将外存中那些重又具备运行条件的就绪进程重新调入内存，修改其状态和挂到就绪进程队列等待进程调度

□ 实质

- 存储器管理中的对换功能

第三章 处理机调度与死锁

3.1 高级、中级与低级调度

3.2 调度队列模型

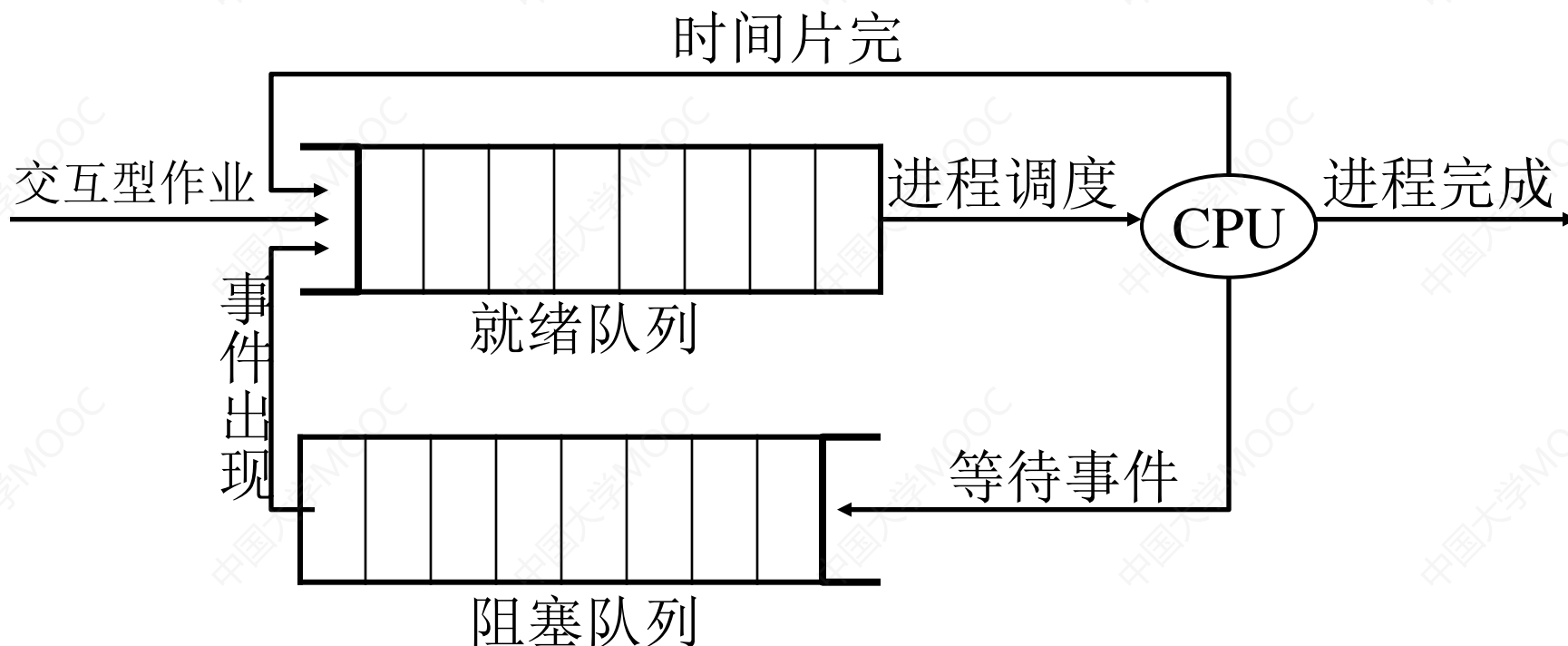
3.3 调度方式与算法选择准则

3.4 调度算法

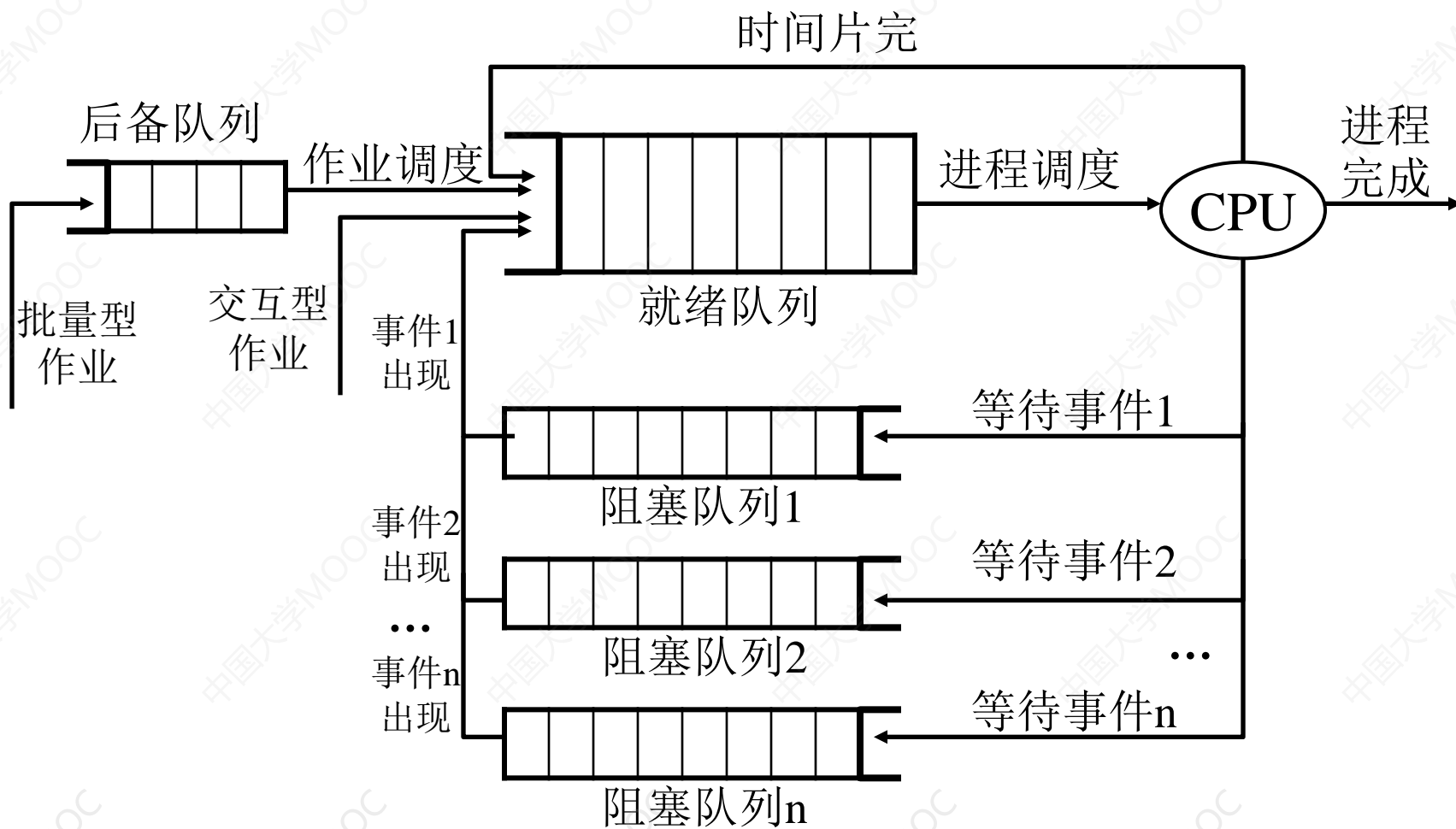
3.5 死锁产生及处理策略

3.6 死锁避免与银行家算法

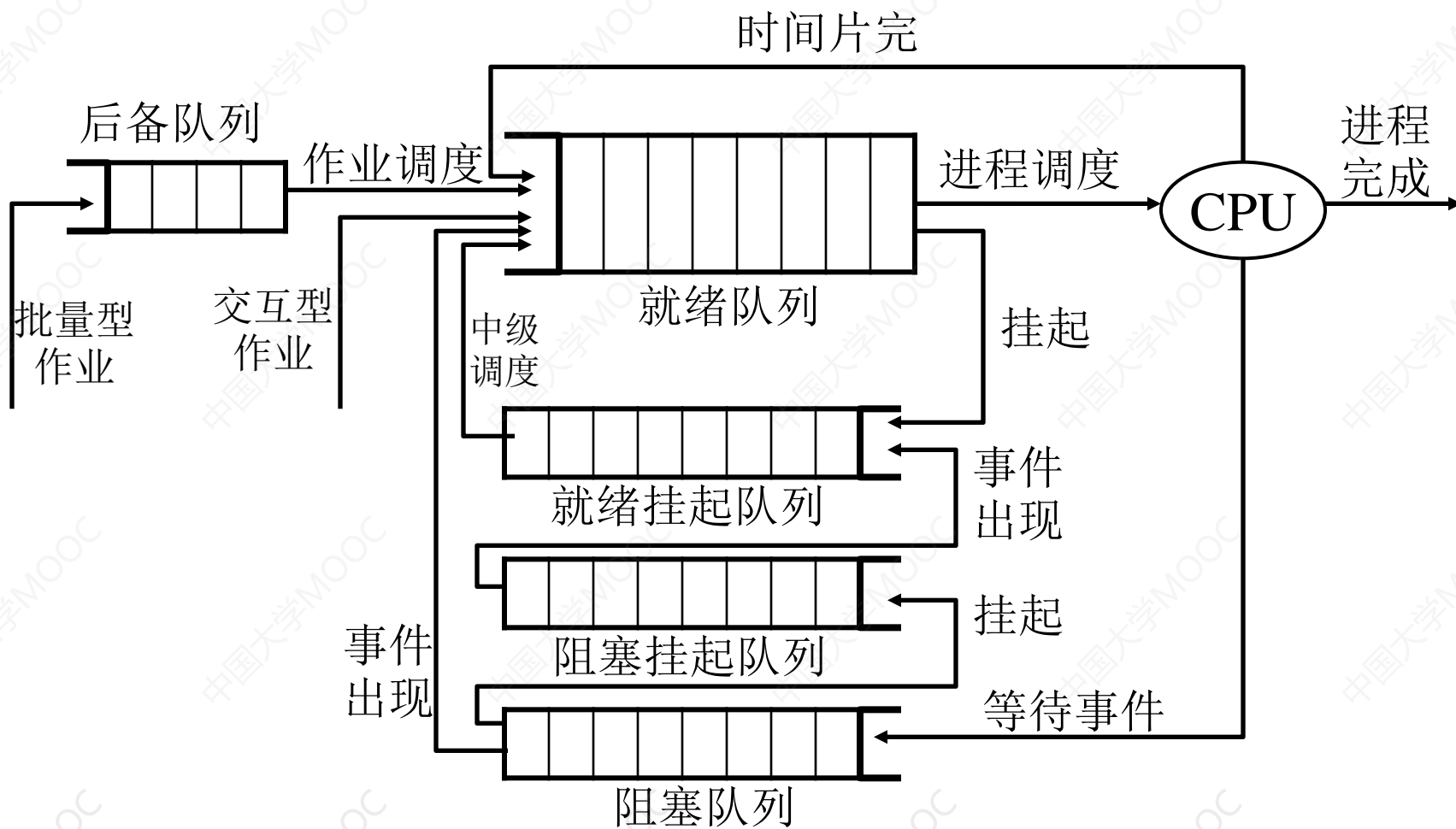
仅有进程调度的调度队列模型



具有高级和低级调度的调度队列模型



同时具有三级调度的调度队列模型



第三章 处理机调度与死锁

3.1 高级、中级与低级调度

3.2 调度队列模型

3.3 调度方式与算法选择准则

3.4 调度算法

3.5 死锁产生及处理策略

3.6 死锁避免与银行家算法

选择调度方式和算法的若干准则

□ 面向用户的准则（与操作系统类型有关）

- 周转时间短（平均周转/带权周转时间）
- 响应时间快
- 截至时间的保证
- 优先权准则

□ 面向系统的准则

- 系统吞吐量高
- 处理机利用率好
- 各类资源的平衡利用

$$T = \frac{1}{n} \left[\sum_{i=1}^n T_i \right]$$

$$W = \frac{1}{n} \left[\sum_{i=1}^n \frac{T_i}{T_{S_i}} \right]$$

第三章 处理机调度与死锁

3.1 高级、中级与低级调度

3.2 调度队列模型

3.3 调度方式与算法选择准则

3.4 调度算法

3.5 死锁产生及处理策略

3.6 死锁避免与银行家算法

3.4 调度算法（资源分配算法）

3.4.1 先来先服务调度算法

3.4.2 短作业（进程）优先调度算法

3.4.3 高优先权优先调度算法

3.4.4 高响应比优先调度算法

3.4.5 时间片轮转调度算法

3.4.6 多级队列调度算法

3.4.7 多级反馈队列调度算法

3.4.8 实时调度算法

先来先服务调度算法举例分析

进程名	到达时刻	服务时间	开始执行时刻	完成时刻	周转时间	带权周转时间
A	0	1				
B	1	100				
C	2	1				
D	3	100				

先来先服务调度算法FCFS

□ 基本思想

- 先来先服务作业调度算法
- 先来先服务进程调度算法

□ 算法特点

- 有利于长作业（进程）而不利于短作业（进程）
- 有利于CPU繁忙型作业（进程）而不利于I/O繁忙型作业（进程）

3.4 调度算法（资源分配算法）

3.4.1 先来先服务调度算法

3.4.2 短作业（进程）优先调度算法

3.4.3 高优先权优先调度算法

3.4.4 高响应比优先调度算法

3.4.5 时间片轮转调度算法

3.4.6 多级队列调度算法

3.4.7 多级反馈队列调度算法

3.4.8 实时调度算法

短作业[进程]优先调度算法举例分析

作业情况 先来先服务 短进程优先	进程名	A	B	C	D	E	平均
	到达时刻	0	1	2	3	4	
	服务时间	4	3	5	2	4	
	完成时刻						
	周转时间						
	带权周转时间						
	完成时刻						
	周转时间						
	带权周转时间						

短作业优先调度算法SJF/SJN

□ 基本思想

- 作业调度：选取服务时间最短的**若干道**作业装入内存，并采用适当进程调度算法调度执行
- 单道批处理系统、多道批处理系统

□ 算法特点

- 能有效降低作业平均等待时间和提高系统吞吐量
- 不利于长作业
- 完全未考虑作业的紧迫程度
- 作业服务时间估计的不准确性

进程运行过程特点及调度考量

□ 进程运行过程分析

- 计算与I/O操作交替发生
- **交互式作业**通常在I/O操作之间仅运行很短时间
- **批处理作业**在I/O操作之间可能运行很长时间

□ 进程调度考量

- 赋予交互式作业以较高优先级的一种方式，是基于**进程的下一轮的处理器集中使用时间（即执行I/O操作之前的时间量，CPU burst）**的长短来确定其相应的优先级
- 根据进程以往执行情况来推测这一时间量

最短运行时间优先调度算法SRTF

□ 基本思想

- 进程调度：选取下一轮处理器集中使用时间（CPU burst）最短的进程，并优先调度执行

□ CPU burst推测公式

$$E_i = (\theta * T_{i-1}) + ((1-\theta) * E_{i-1})$$

$\theta \in [0, 1]$ ，可初始化为0.5

□ 算法特点

- 优先调度执行交互式作业，有助于改善用户体验
- 未考虑进程的紧迫程度

3.4 调度算法（资源分配算法）

3.4.1 先来先服务调度算法

3.4.2 短作业（进程）优先调度算法

3.4.3 高优先权优先调度算法

3.4.4 高响应比优先调度算法

3.4.5 时间片轮转调度算法

3.4.6 多级队列调度算法

3.4.7 多级反馈队列调度算法

3.4.8 实时调度算法

高优先权优先调度算法FPF

❑ 基本思想

- 照顾紧迫型作业（进程）

基于进程/用户
公平的调度算法

❑ 算法分类

- 非抢占式优先权算法
- 抢占式优先权调度算法

❑ 优先权类型

- 静态优先权
- 动态优先权

进程优先权确
定依据：进程
类型、资源需
求及用户要求

3.4 调度算法（资源分配算法）

3.4.1 先来先服务调度算法

3.4.2 短作业（进程）优先调度算法

3.4.3 高优先权优先调度算法

3.4.4 高响应比优先调度算法

3.4.5 时间片轮转调度算法

3.4.6 多级队列调度算法

3.4.7 多级反馈队列调度算法

3.4.8 实时调度算法

高响应比优先调度算法

□ 基本思想

- 短作业优先调度算法+动态优先权机制

□ 优先权（响应比 R_p ）

- （等待时间+要求服务时间）/要求服务时间

□ 算法特点

- 短作业与先后次序的兼顾，且不会使长作业长期得不到服务
- 响应比计算系统开销

3.4 调度算法（资源分配算法）

3.4.1 先来先服务调度算法

3.4.2 短作业（进程）优先调度算法

3.4.3 高优先权优先调度算法

3.4.4 高响应比优先调度算法

3.4.5 时间片轮转调度算法

3.4.6 多级队列调度算法

3.4.7 多级反馈队列调度算法

3.4.8 实时调度算法

时间片轮转调度算法

□ 基本思想

- 按先来先服务原则排队
- 时间片及时钟中断

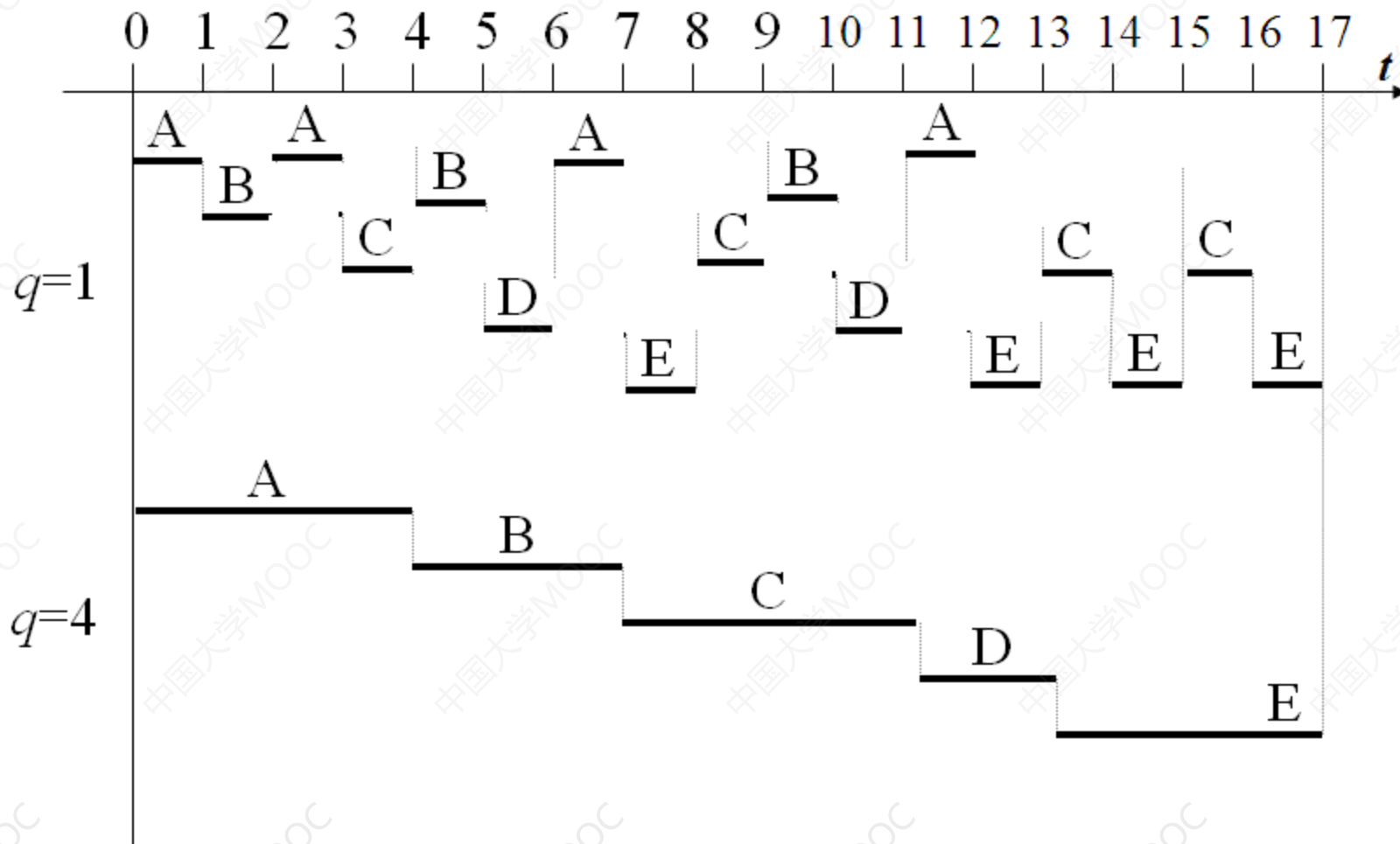
□ 时间片大小的确定

- 系统对响应时间的要求
- 就绪队列中进程的数目
- 系统的处理能力

时间片轮转调度算法举例分析 待续

进程情况	进程名	A	B	C	D	E	平均	进程就绪队列 5时刻
	到达时刻	0	1	2	3	4		
	服务时间	4	3	4	2	4		
$q=1$	完成时刻							D
	周转时间							A
	带权周转时间							E
$q=4$	完成时间							C
	周转时刻							B
	带权周转时间							^

时间片轮转调度算法举例分析 待续



时间片轮转调度算法举例分析 续完

进程情况	进程名	A	B	C	D	E	平均
	到达时刻	0	1	2	3	4	
	服务时间	4	3	4	2	4	
$q=1$	完成时刻	12	10	16	11	17	
	周转时间	12	9	14	8	13	11.2
	带权周转时间	3	3	3.5	4	3.25	3.35
$q=4$	完成时刻	4	7	11	13	17	
	周转时间	4	6	9	10	13	8.4
	带权周转时间	1	2	2.25	5	3.25	2.7

3.4 调度算法（资源分配算法）

3.4.1 先来先服务调度算法

3.4.2 短作业（进程）优先调度算法

3.4.3 高优先权优先调度算法

3.4.4 高响应比优先调度算法

3.4.5 时间片轮转调度算法

3.4.6 多级队列调度算法

3.4.7 多级反馈队列调度算法

3.4.8 实时调度算法

多级队列调度算法

□ 引入的必要性

- 多操作系统类型配置
- 批量型作业和交互性作业性质的不同

□ 基本思想

- 作业性质分类排列，不同队列不同调度算法

□ 队列间关系处理

- 优先权方式
- 前/后台比例方式

3.4 调度算法（资源分配算法）

3.4.1 先来先服务调度算法

3.4.2 短作业（进程）优先调度算法

3.4.3 高优先权优先调度算法

3.4.4 高响应比优先调度算法

3.4.5 时间片轮转调度算法

3.4.6 多级队列调度算法

3.4.7 多级反馈队列调度算法

3.4.8 实时调度算法

多级反馈队列调度算法

□ 引入的必要性

- 各类进程调度算法均有一定的局限性

□ 基本思想

- 设置多个就绪队列并赋予各个队列不同优先权
- 不同队列中进程执行的时间片大小各不相同
- 先来先服务调度算法与时间片轮转调度算法相结合
- 队列间调度准则和抢占式优先权调度

□ 算法性能

- 能较好地满足各种类型用户（终端型作业用户、短批处理作业用户、长批处理作业用户）的要求

3.4 调度算法（资源分配算法）

3.4.1 先来先服务调度算法

3.4.2 短作业（进程）优先调度算法

3.4.3 高优先权优先调度算法

3.4.4 高响应比优先调度算法


3.4.5 时间片轮转调度算法

3.4.6 多级队列调度算法

3.4.7 多级反馈队列调度算法

3.4.8 实时调度算法

实时调度算法分类



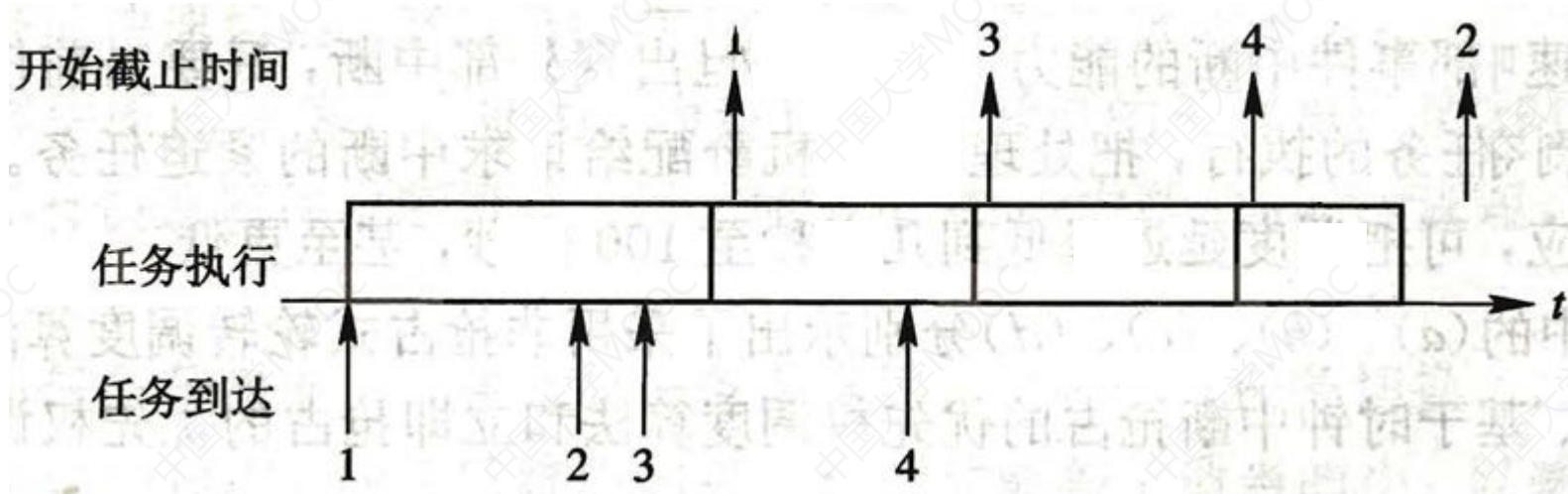
优先级倒置
及解决方案

- 根据实时任务性质的不同
 - 硬实时调度算法和软实时调度算法
- 根据调度方式的不同
 - 非抢占式调度算法和抢占式调度算法
- 根据调度面向实时任务组类型的不同
 - 静态调度算法和动态调度算法
- 基本调度策略分类
 - 时间片轮转调度算法和优先级调度算法
- 多处理机环境
 - 集中式调度和分布式调度

常用实时调度算法1

□ 最早截止时间优先调度算法

➤ EDF (Earliest Deadline First)



常用实时调度算法1

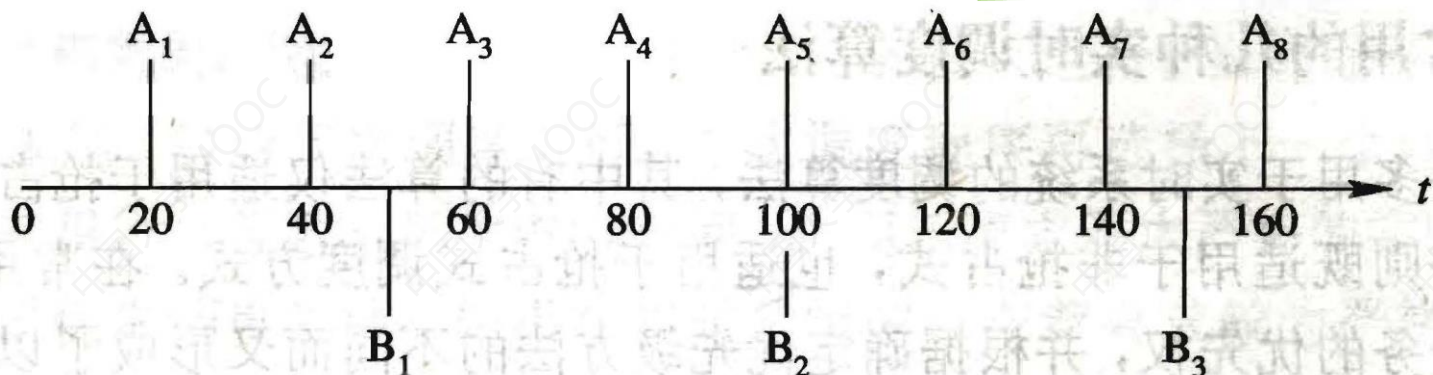
□ 最早截止时间优先调度算法

➤ EDF (Earliest Deadline First)

周期 / 执行时间

A_i : 20ms/10ms

B_j : 50ms/25ms



周期性实时任务示例调度过程:

0ms时刻 (**A1**、B1) : A1; 10ms时刻 (B1) : B1[10ms];

20ms时刻 (**A2**、B1) : A2; 30ms时刻 (B1) : B1[**15ms**];

45ms时刻 (**A3**) : A3; 55ms时刻 (B2) : B2[5ms];

常用实时调度算法2

最低松弛度优先调度算法

周期 / 执行时间
 A_i : 20ms/10ms
 B_j : 50ms/25ms

LLF (Least Laxity First)

松弛度 = 任务结束截止时间 - 当前时间 - 剩余执行时间

0ms时刻松弛度

$L_{A1} = 10\text{ms}$

$L_{B1} = 45\text{ms}$

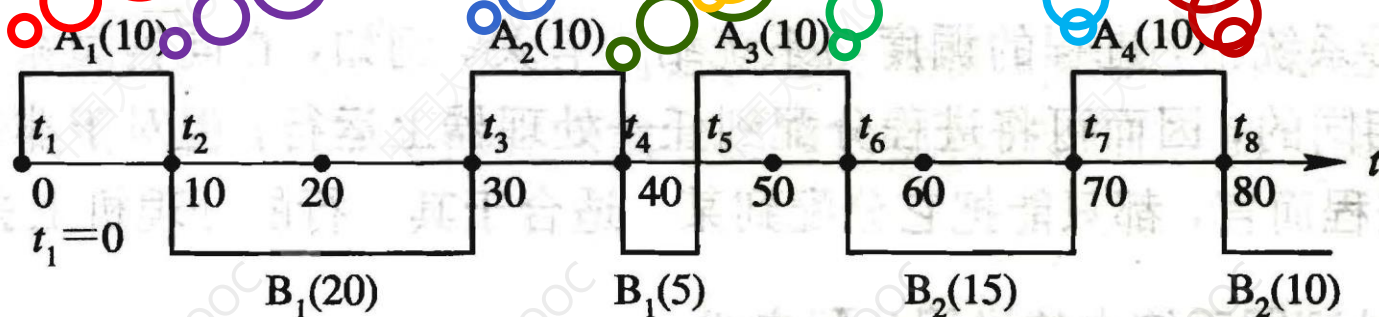
下一调度时刻

80ms时刻松弛度:

$L_{A5} = 100 - 80 - 10 = 10\text{ms}$

$L_{B2} = 100 - 80 - 10 = 10\text{ms}$

下一调度时刻 $\min\{80+10, 100-10\}$



3.4 调度算法（资源分配算法）

3.4.1 先来先服务调度算法

3.4.2 短作业（进程）优先调度算法

3.4.3 高优先权优先调度算法

3.4.4 高响应比优先调度算法

3.4.5 时间片轮转调度算法

3.4.6 多级队列调度算法

3.4.7 多级反馈队列调度算法

3.4.8 实时调度算法

作业题

- **3.1** 谈谈你对处理机调度模型及各级调度机制的认识与理解。
- **3.2** 选择调度方式和调度算法时，应遵循哪些准则？并请分析比较各种调度算法的基本思想、关键要领、优缺点及适用场合。

实验课题

□ 实验课题7

处理器调度算法模拟实现与比较

实验课题

□ 实验课题8

Linux处理器调度机制及相关调度算法探析

实验课题

□ 实验课题9

Linux处理器调度新型算法设计实现 与测试验证

第三章 处理机调度与死锁

3.1 高级、中级与低级调度

3.2 调度队列模型

3.3 调度方式与算法选择准则

3.4 调度算法

3.5 死锁产生及处理策略

3.6 死锁避免与银行家算法

3.5 死锁产生及处理策略

3.5.1 死锁的基本概念

3.5.2 死锁产生的原因

3.5.3 死锁产生的必要条件

3.5.4 处理死锁的基本方法

3.5.5 死锁的预防

3.5.6 死锁的检测与解除


死锁的基本概念

□ 死锁 (Deadlock)

- 在多道程序系统中，并发执行的多个进程因争夺资源而造成的一种若无外力作用有关进程都将永远不能向前推进的僵持状态或僵局

□ 资源分类

- 可剥夺资源与不可剥夺资源
- 可重用资源与消耗性资源。



永久性资源与临时性资源

3.5 死锁产生及处理策略

3.5.1 死锁的基本概念

3.5.2 死锁产生的原因

3.5.3 死锁产生的必要条件

3.5.4 处理死锁的基本方法

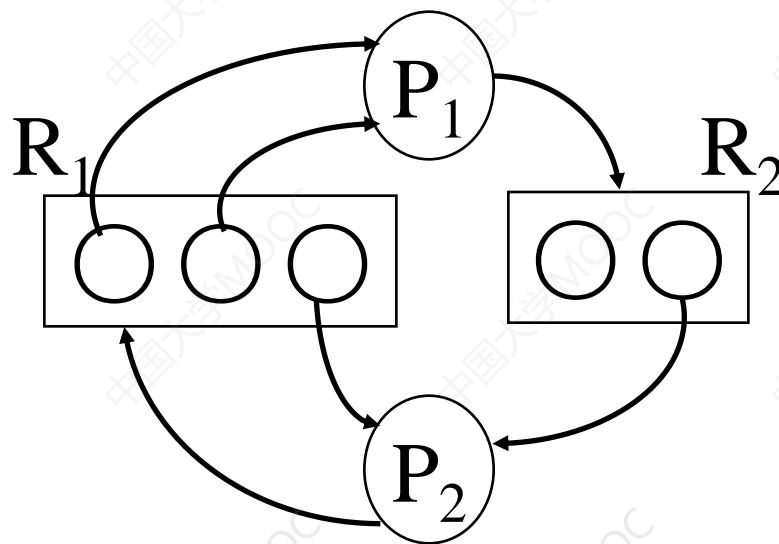
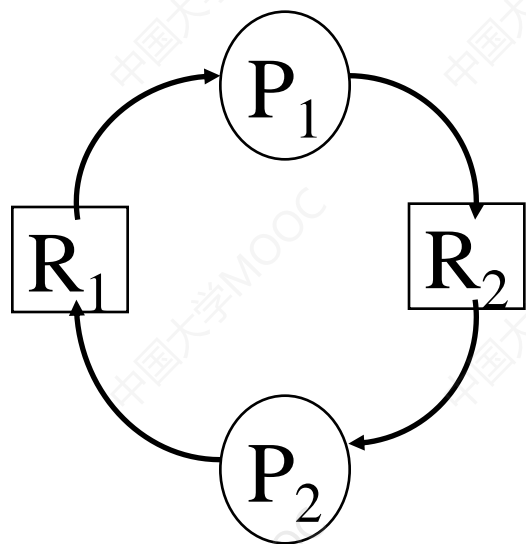
3.5.5 死锁的预防

3.5.6 死锁的检测与解除

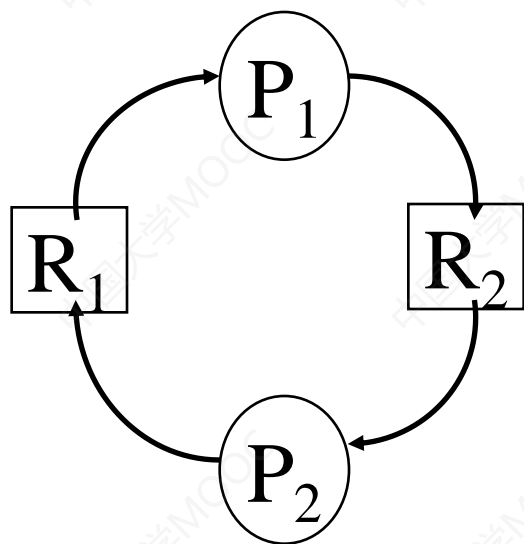
资源分配图

□ $G=(P \cup R, E)$

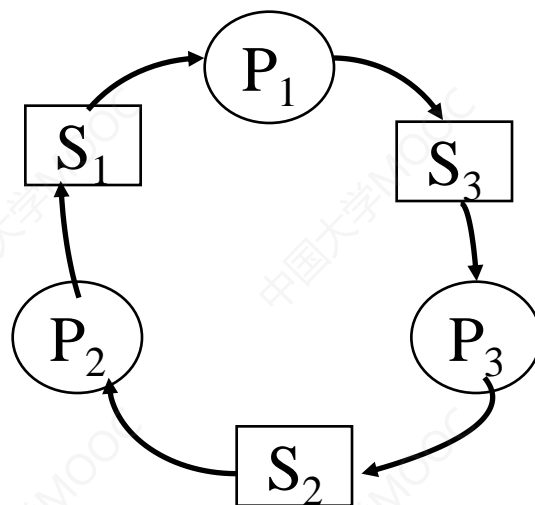
- 资源请求边 $e = \langle P_i, R_j \rangle$
- 资源分配边 $e = \langle R_j, P_i \rangle$



死锁产生原因之一：竞争资源

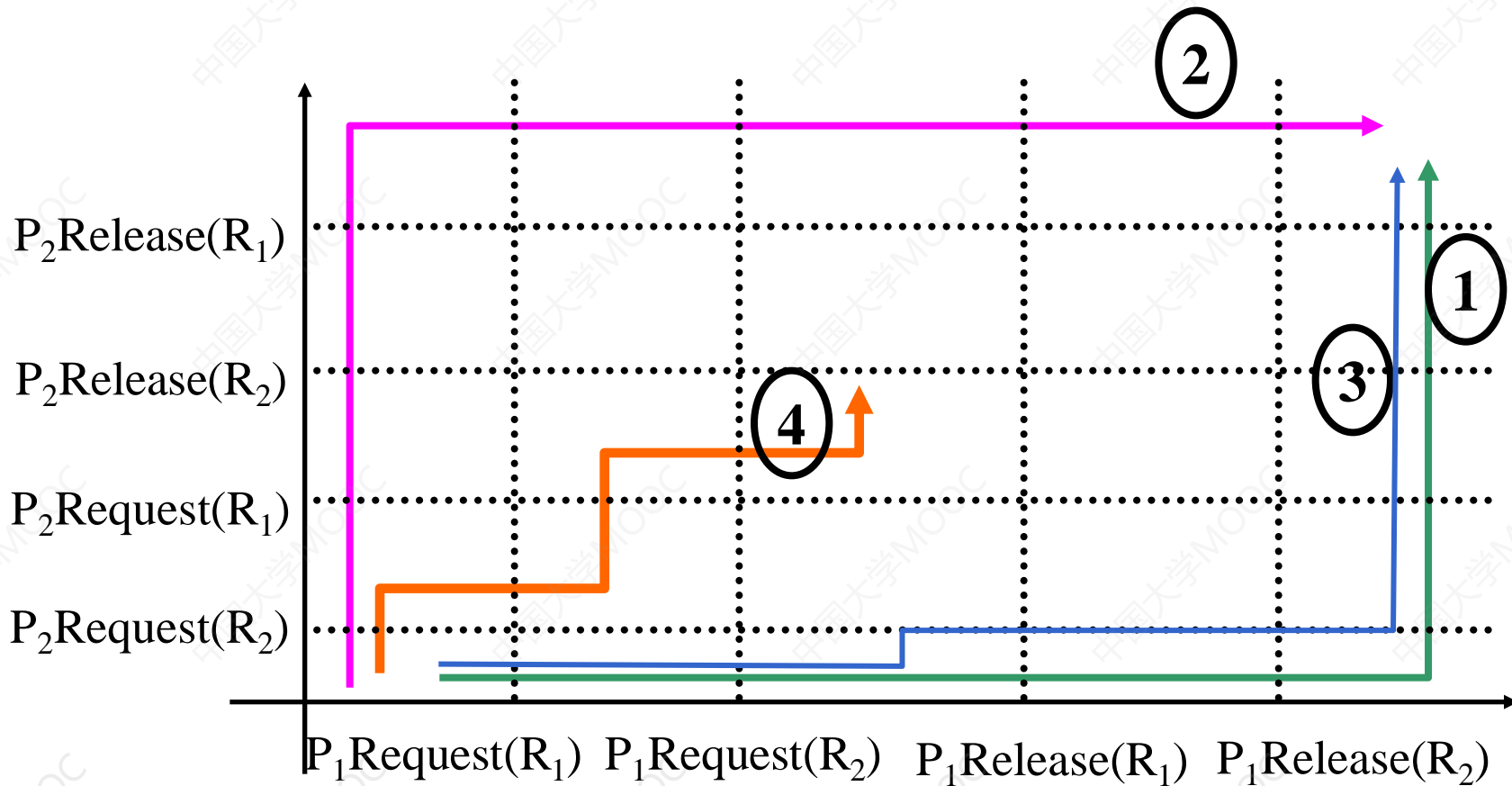


I/O设备共享时的死锁情况
(A) 竞争不可剥夺性资源



进程间通信时的死锁情况
(B) 竞争临时性资源

死锁产生原因之二：进程推进次序“非法”



3.5 死锁产生及处理策略

3.5.1 死锁的基本概念

3.5.2 死锁产生的原因

3.5.3 死锁产生的必要条件

3.5.4 处理死锁的基本方法

3.5.5 死锁的预防

3.5.6 死锁的检测与解除

产生死锁的必要条件

- ❑ 互斥条件
 - 资源排它性使用
- ❑ 请求和保持条件
 - 请求资源未果进程虽阻塞但保持占有资源不放
- ❑ 不剥夺条件
 - 进程已获资源未使用完之前不能被剥夺
- ❑ 环路等待条件
 - 进程-资源环形链 $\{P_0, P_1, P_2, \dots, P_n\}$

3.5 死锁产生及处理策略

3.5.1 死锁的基本概念

3.5.2 死锁产生的原因

3.5.3 死锁产生的必要条件

3.5.4 处理死锁的基本方法

3.5.5 死锁的预防

3.5.6 死锁的检测与解除

处理死锁的基本方法

□ 预防死锁

- 设置某些限制前提以破坏产生死锁必要条件

□ 避免死锁

- 资源动态分配过程中，利用某种方法去防止系统进入不安全状态

□ 检测死锁

- 运行过程中通过系统设置的检测机构及时检测死锁的发生，并精确确定相关进程和资源

□ 解除死锁

- 撤销或挂起一些进程以回收资源和再分配

3.5 死锁产生及处理策略

3.5.1 死锁的基本概念

3.5.2 死锁产生的原因

3.5.3 死锁产生的必要条件

3.5.4 处理死锁的基本方法

3.5.5 死锁的预防

3.5.6 死锁的检测与解除

死锁的预防策略之一

□ 摒弃“请求和保持”条件 ⇔ 一次性申请

- 系统要求所有进程在开始运行之前，都必须一次性地申请其在整个运行过程所需的全部资源和进行一次分配
- 简单、安全且易于实现
- 资源浪费、进程延迟

死锁的预防策略之二

❑ 摒弃“不剥夺”条件 ⇔ 主动释放

- 进程在需要资源时才提出请求，且得不到满足时应释放其已占有资源
- 实现复杂，代价很大（反复地申请与释放资源、进程周转时间延长、系统吞吐量降低、系统开销增加）

死锁的预防策略之三

❑ 摒弃“环路等待”条件 ⇔ 申请有序

- 所有资源按类型进行线性排队，所有进程对资源的请求严格按资源序号递增次序提出
- 资源次序的不灵活性（新设备、程序逻辑设计与编程限制及资源浪费）

3.5 死锁产生及处理策略

3.5.1 死锁的基本概念

3.5.2 死锁产生的原因

3.5.3 死锁产生的必要条件

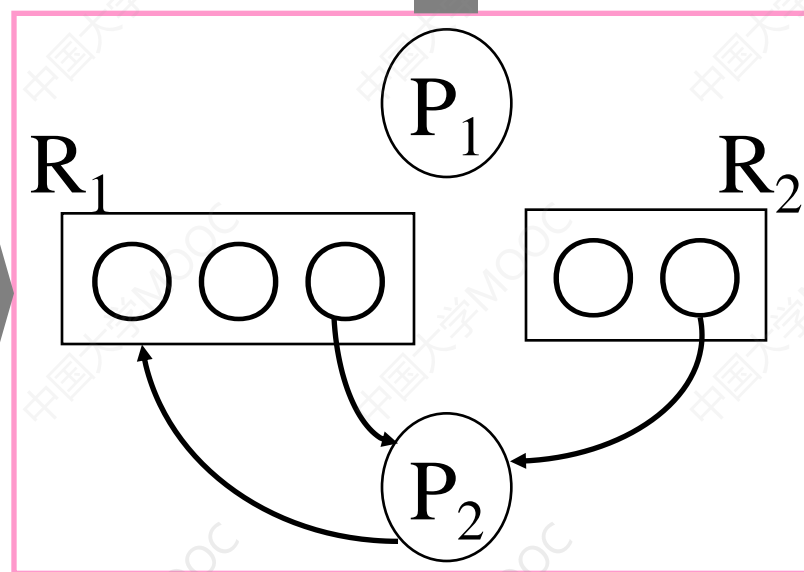
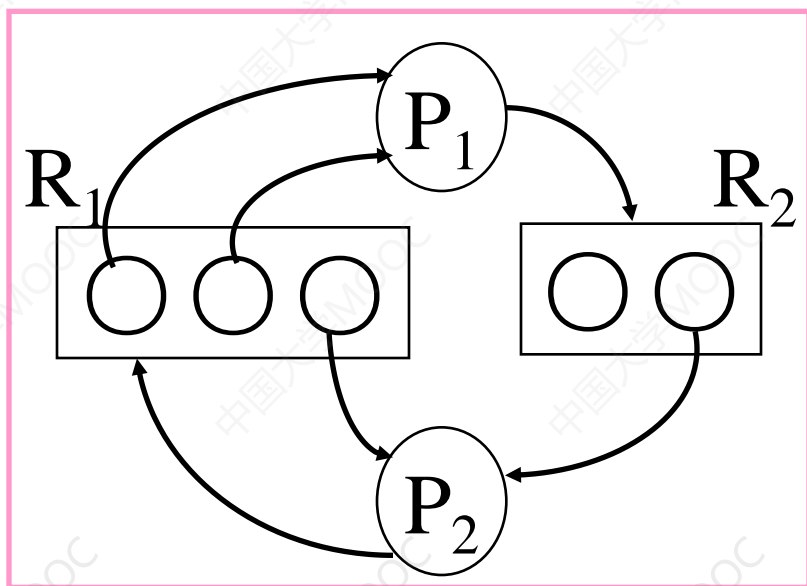
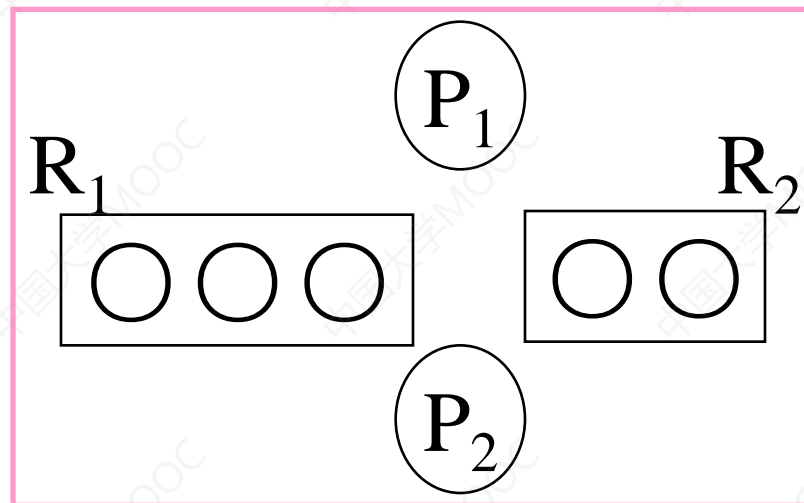
3.5.4 处理死锁的基本方法

3.5.5 死锁的预防

3.5.6 死锁的检测与解除

死锁定理

- ❑ 系统状态S为死锁状态的充要条件是当且仅当该状态下的资源分配图是不可完全化简的



死锁检测算法

1 令 $Work = Available$

$IsolatedSet = \{P_i \mid Allocation[i] = \mathbf{0} \wedge Request_i = \mathbf{0}\}$

2 **for** all $P_i \notin IsolatedSet$ **do**

begin

for all $Request_i \leq Work$ **do**

begin

$Work = Work + Allocation[i];$

$IsolatedSet = IsolatedSet \cup \{P_i\}$

end

end

3 $deadlock = \sim (IsolatedSet == \{P_1, P_2, \dots, P_n\});$

死锁检测算法ZGS版

```
1 令  $IsolatedSet = \{P_i \mid Allocation[i] = 0 \wedge Request_i = 0\}$ ;  
    $SetToSolve = \{P_1, P_2, \dots, P_n\} - IsolatedSet$ ;  
    $Work = Available$ ;  $zSetToSolve = SetToSolve$ ;  
2 while ( $SetToSolve \neq NULL$ )  
  { for ( $P_i \in SetToSolve$ )  
    { if ( $Request_i \leq Work$ )  
      {  $Work = Work + Allocation[i]$ ;  
         $SetToSolve = SetToSolve - \{P_i\}$ ; } }  
    if ( $zSetToSolve == SetToSolve$ )  
      break;  
    else  $zSetToSolve = SetToSolve$ ; }  
3  $deadlock = (SetToSolve \neq NULL)$ ;
```

死锁的解除

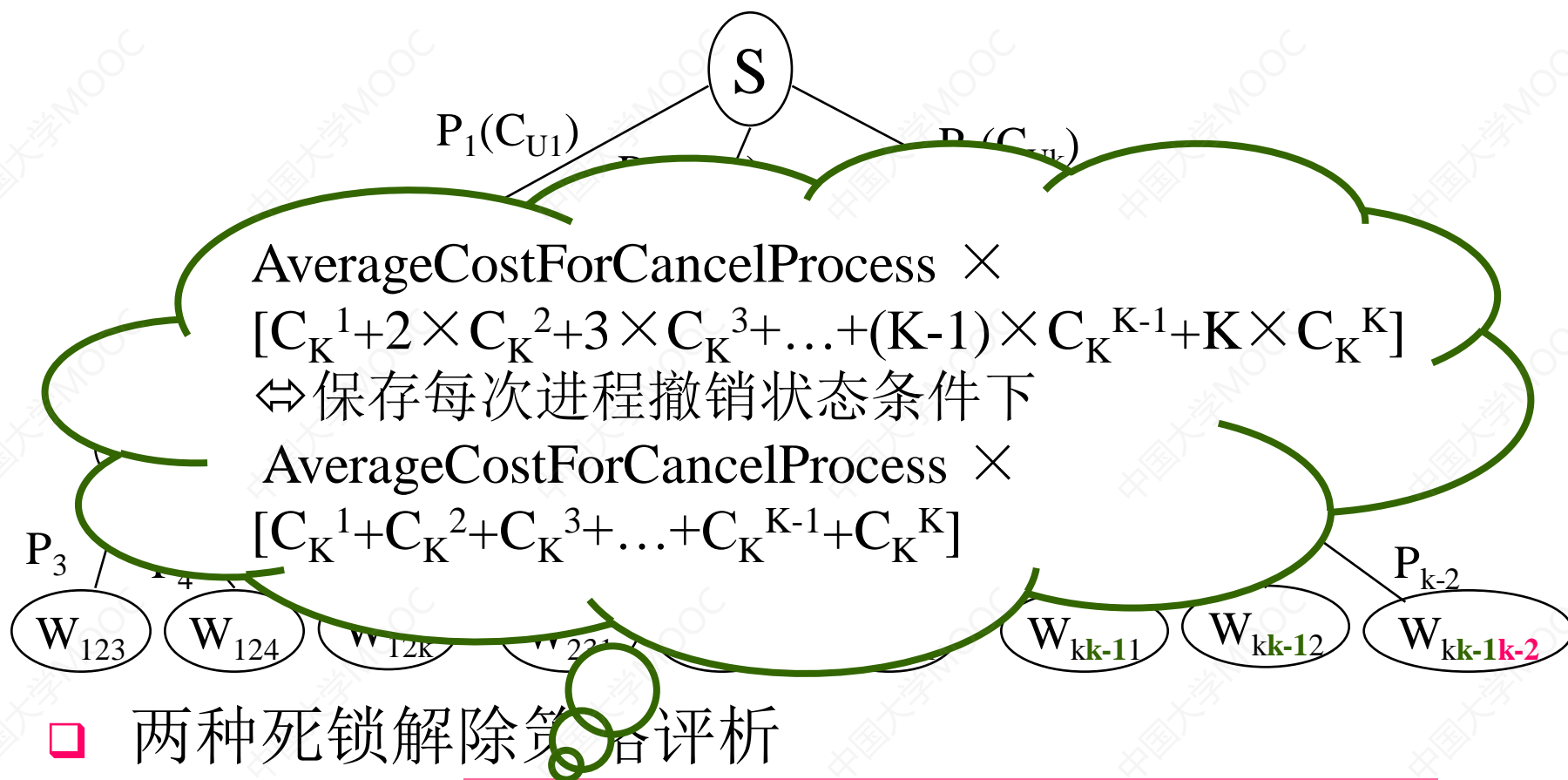
□ 基本方法

- 剥夺其它进程足够数量资源给死锁进程
- 撤消死锁进程（全部撤销或逐个撤销）

□ 死锁解除策略及评价指标

- 为解除死锁所需撤消的进程数目最少
- 立足于撤消死锁进程所付出的代价最小

死锁解除策略实例评析



两种死锁解除策略评析

- $Cost_{Max} = \text{AverageCostForCancelProcess} \times (2^K - 1)$
- $Cost_{Max} = \sum \min \{ \text{CostForCancelProcess}_i \}$

3.5 死锁产生及处理策略

3.5.1 死锁的基本概念

3.5.2 死锁产生的原因

3.5.3 死锁产生的必要条件

3.5.4 处理死锁的基本方法

3.5.5 死锁的预防

3.5.6 死锁的检测与解除

第三章 处理机调度与死锁

3.1 高级、中级与低级调度

3.2 调度队列模型

3.3 调度方式与算法选择准则

3.4 调度算法

3.5 死锁产生及处理策略

3.6 死锁避免与银行家算法

死锁避免

□ 基本思想

- 允许进程动态地申请资源，但系统在进行资源分配之前，应首先就资源分配的安全性进行检查，且仅当确认此次分配不会导致系统进入不安全状态时才可分配，否则予以拒绝

死锁避免基本概念

□ 安全状态

- 指系统可按某种进程序列 $\langle P_1, P_2, \dots, P_n \rangle$ （称之为安全分配序列）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，使每个进程均能顺利完成

□ 不安全状态

- 系统无法找到一个安全分配序列的系统状态

□ 死锁与状态安全性之间的关系

- 并非所有不安全状态都是死锁状态
- 当系统进入不安全状态后，便可能陷入死锁
- 只要保证系统处于安全状态，便可避免死锁

安全状态及向不安全状态的转换

资源名称	资源总数	可用资源量
磁带机	12	
进程	最大需求	已分配(尚需)
P_1	10	5 (5)
P_2	4	2 (2)
P_3	9	2 (7)

- T_0 时刻存在安全分配序列 $\langle P_2, P_1, P_3 \rangle$
- 若在 T_0 时刻应进程请求将所剩磁带机中的1台分配给 P_3 ，则系统进入不安全状态

银行家算法之数据结构

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3			
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			

- 可利用资源总量 $Available[A,B,C] = \{10, 5, 7\}$
- T_0 时刻请求向量 $Request_4[A,B,C] = \{3, 3, 1\}$
- 工作向量 [m]，进程完成布尔向量 [n]

银行家算法之数据结构

- 可利用资源向量/请求向量[m]
 - $Available[j]=k$ 表示系统现有 k 个 R_j 类资源
 - $Request_i[j]=k$ 表示进程 P_i 请求 k 个 R_j 类资源
- 最大需求矩阵/分配矩阵/需求矩阵[n, m]
 - $Max[i, j]=k$ 表示进程 P_i 最多需要 k 个 R_j 类资源
 - $Allocation[i, j]=k$ 表示进程 P_i 已有 k 个 R_j 类资源
 - $Need[i, j]=k$ 表示进程 P_i 尚需 k 个 R_j 类资源
- 工作向量 [m] / Finish布尔向量 [n]
 - $Work[j]=k$ 表示系统“可”提供 k 个 R_j 类资源
 - $Finish[i]$ 表示进程 P_i 可否拥有足够资源完成运行

银行家算法之主体算法

- 1 进程 P_i 发出资源请求 $Request_i$
- 2 若非 $Request_i \leq Need[i]$, 出错返回
- 3 若非 $Request_i \leq Available$, 则应使 P_i 等待并返回
- 4 系统试探性地满足 P_i 请求, 并作以下修改:
 - $Available = Available - Request_i$
 - $Allocation[i] = Allocation[i] + Request_i$
 - $Need[i] = Need[i] - Request_i$
- 5 系统调用安全性算法进行资源分配检查, 若安全则执行分配, 否则恢复试探分配前状态, 并使 P_i 等待

银行家算法之安全性子算法

- 1 令 $Work = Available$, $Finish = FALSE$
- 2 从进程集合中查找一个满足 $Finish[i] = FALSE$ 且 $Need[i] \leq Work$ 的进程 P_i 。若找到, 则可假定 P_i 能获得所需资源并顺利执行, 故有:
 - $Work = Work + Allocation[i]$
 - $Finish[i] = True$然后重复执行第2步; 否则转至第3步执行
- 3 如果 $Finish = TRUE$, 则表示系统处于安全状态; 否则系统处于不安全状态

银行家算法应用举例之一_A(待续)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3			
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			

- ❑ 系统资源总量 $Available[A,B,C] = \{10, 5, 7\}$
- ❑ T_0 时刻 $Available[A,B,C] = \{3, 3, 2\}$
- ❑ 安全分配序列 ?

银行家算法应用举例之一B(待续)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3			
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			

- ❑ 系统资源总量 $Available[A, B, C] = \{10, 5, 7\}$
- ❑ T_0 时刻 $Available[A, B, C] = \{3, 3, 2\}$
- ❑ 安全分配序列 ?

银行家算法应用举例之一C(待续)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3			
P ₁	3 2 2	2 0 0	1 2 2	3 3 2	5 3 2	① True
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			

- ❑ 系统资源总量 $Available[A,B,C] = \{10, 5, 7\}$
- ❑ T_0 时刻 $Available[A,B,C] = \{3, 3, 2\}$
- ❑ 安全分配序列 ?

银行家算法应用举例之一D(待续)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3			
P ₁	3 2 2	2 0 0	1 2 2	3 3 2	5 3 2	① True
P ₂	9 0 2	3 0 2	6 0 0			
P ₃	2 2 2	2 1 1	0 1 1	5 3 2	7 4 3	② True
P ₄	4 3 3	0 0 2	4 3 1			

- ❑ 系统资源总量 $Available[A,B,C] = \{10, 5, 7\}$
- ❑ T_0 时刻 $Available[A,B,C] = \{3, 3, 2\}$
- ❑ 安全分配序列 ?

银行家算法应用举例之一E(待续)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3			
P ₁	3 2 2	2 0 0	1 2 2	3 3 2	5 3 2	① True
P ₂	9 0 2	3 0 2	6 0 0			
P ₃	2 2 2	2 1 1	0 1 1	5 3 2	7 4 3	② True
P ₄	4 3 3	0 0 2	4 3 1	7 4 3	7 4 5	③ True

- ❑ 系统资源总量 $Available[A,B,C] = \{10, 5, 7\}$
- ❑ T_0 时刻 $Available[A,B,C] = \{3, 3, 2\}$
- ❑ 安全分配序列 ?

银行家算法应用举例之一F(待续)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3	7 4 5	7 5 5	④ True
P ₁	3 2 2	2 0 0	1 2 2	3 3 2	5 3 2	① True
P ₂	9 0 2	3 0 2	6 0 0			
P ₃	2 2 2	2 1 1	0 1 1	5 3 2	7 4 3	② True
P ₄	4 3 3	0 0 2	4 3 1	7 4 3	7 4 5	③ True

- ❑ 系统资源总量 $Available[A,B,C] = \{10, 5, 7\}$
- ❑ T_0 时刻 $Available[A,B,C] = \{3, 3, 2\}$
- ❑ 安全分配序列 $\langle P_1, P_3, P_4, P_0, P_2 \rangle$

银行家算法应用举例之二_A(待续)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3			
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			

- T_1 时刻 $Available[A,B,C] = \{3, 3, 2\}$
- 进程P₁发出资源请求 $Request_1(1,0,2) < Need_1(1,2,2)$
- 安全分配序列 ?

银行家算法应用举例之二B(待续)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3			
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			

- 进程P₁发出资源请求 $Request_1(1,0,2) < Need_1(1,2,2)$
- T_1 时刻 $Available[A,B,C] = \{3, 3, 2\} \Rightarrow \{2, 3, 0\}$
- 安全分配序列 ?

银行家算法应用举例之二c(待续)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3			
P ₁	3 2 2	3 0 2	0 2 0	2 3 0	5 3 2	① True
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			

- 进程P₁发出资源请求 $Request_1(1,0,2) < Need_1(1,2,2)$
- T_1 时刻 $Available[A,B,C] = \{3, 3, 2\} \Rightarrow \{2, 3, 0\}$
- 安全分配序列 ?

银行家算法应用举例之二D(待续)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3			
P ₁	3 2 2	3 0 2	0 2 0	2 3 0	5 3 2	① True
P ₂	9 0 2	3 0 2	6 0 0			
P ₃	2 2 2	2 1 1	0 1 1	5 3 2	7 4 3	② True
P ₄	4 3 3	0 0 2	4 3 1			

- 进程P₁发出资源请求 $Request_1(1,0,2) < Need_1(1,2,2)$
- T_1 时刻 $Available[A,B,C] = \{3, 3, 2\} \Rightarrow \{2, 3, 0\}$
- 安全分配序列 ?

银行家算法应用举例之二E(待续)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3			
P ₁	3 2 2	3 0 2	0 2 0	2 3 0	5 3 2	① True
P ₂	9 0 2	3 0 2	6 0 0			
P ₃	2 2 2	2 1 1	0 1 1	5 3 2	7 4 3	② True
P ₄	4 3 3	0 0 2	4 3 1	7 4 3	7 4 5	③ True

- 进程P₁发出资源请求 $Request_1(1,0,2) < Need_1(1,2,2)$
- T_1 时刻 $Available[A,B,C] = \{3, 3, 2\} \Rightarrow \{2, 3, 0\}$
- 安全分配序列 ?

银行家算法应用举例之二F(待续)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3	7 4 5	7 5 5	④ True
P ₁	3 2 2	3 0 2	0 2 0	2 3 0	5 3 2	① True
P ₂	9 0 2	3 0 2	6 0 0			
P ₃	2 2 2	2 1 1	0 1 1	5 3 2	7 4 3	② True
P ₄	4 3 3	0 0 2	4 3 1	7 4 3	7 4 5	③ True

- 进程P₁发出资源请求 $Request_1(1,0,2) < Need_1(1,2,2)$
- T_1 时刻 $Available[A,B,C] = \{3, 3, 2\} \Rightarrow \{2, 3, 0\}$
- 安全分配序列 $\langle P_1, P_3, P_4, P_0, P_2 \rangle$

银行家算法应用举例之三(待续)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 1 0	7 4 3			
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			

- T_2 时刻 $Available[A,B,C] = \{2, 3, 0\}$
- 进程P₄发出资源请求 $Request_4(3,3,0) < Need_4(4,3,1)$
- $Request_4(3,3,0) > Available(2,3,0)$, 故让P₄等待

银行家算法应用举例之四(续完)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 3 0	7 2 3			
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			

- T_3 时刻 $Available[A,B,C] = \{2, 3, 0\}$
- 进程P₀发出资源请求 $Request_0(0,2,0) < Need_0(7,4,3)$
- 尝试分配

第三章 处理机调度与死锁

3.1 高级、中级与低级调度

3.2 调度队列模型

3.3 调度方式与算法选择准则

3.4 调度算法

3.5 死锁产生及处理策略

3.6 死锁避免与银行家算法

作业题

- 3.3 谈谈你对死锁的概念、产生原因及其必要条件的认识与理解。同时并就死锁的各种处理策略展开讨论。
- 3.4[必做] 在课本关于银行家算法的例子中，如果P0发出的请求量由Request0 (0,2,0)改为Request0 (0,1,0)，问系统可否将资源分配给它？

实验课题

□ 实验课题10

银行家算法模拟实现

（选择本课题，则不容许选择“死锁检测算法模拟实现”）

实验课题

□ 实验课题11

死锁检测算法模拟实现

（选择本课题，则不容许选择“银行家算法模拟实现”）

第三章

部分习题讲解

作业题

- **3.3** 谈谈你对死锁的概念、产生原因及其必要条件的认识与理解。同时并就死锁的各种处理策略展开讨论。
- **3.4[必做]** 在课本关于银行家算法的例子中，如果P0发出的请求量由Request0 (0,2,0)改为Request0 (0,1,0)，问系统可否将资源分配给它？

银行家算法应用举例之四(续完)

进程	MAX <u>A B C</u>	Allocation <u>A B C</u>	Need <u>A B C</u>	Work <u>A B C</u>	Allocation + Work <u>A B C</u>	<u>Finish</u>
P ₀	7 5 3	0 2 0	7 3 3			
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			

- T_3 时刻 $Available[A,B,C] = \{2, 3, 0\}$
- 进程P₀发出资源请求 $Request_0(0,1,0) < Need_0(7,4,3)$
- 尝试分配:



**同学们，
再见！**

2021年4月27日星期二