操作系统

Xia Tian

Renmin University of China

April 9, 2018

CH3 处理机的调度与死锁

- 3.1 处理机调度的基本概念
- 3.2 调度算法
- 3.3 实时调度
- 3.4 多处理机系统中的调度
- 3.5 产生死锁的原因和必要条件
- 3.6 预防死锁的办法
- 3.7 死锁的检测与解除

进程一章的补充阅读作业

协程介绍：介绍协程的历史，现在和未来，协程与进程、线程的差异等

所有人整理的文档下周一前发助教。

下周开始小组汇报（学习委员按照人数分成 10 个小组）

3.1 处理机调度的基本概念

- 进程调度的定义和目标
- 高级、中级和低级调度
- 调度队列模型
- 选择调度方式和调度算法的若干准则

进程调度的定义

- 单 CPU 环境下，任一时刻只能有一个进程处于执行状态，如何确定在某一时刻由哪个进程执行?
- 调度:
  - 根据特定目标选择下一个要运行的进程
  - 调度的级别或者说粒度问题?

高级、中级和低级调度

- 高级调度
  - 作业调度、长程调度、接纳调度
- 中级调度
- 低级调度
  - 进程调度，短程调度

- 运行频率：低 ∽ 中 ∽ 高

高级调度

- 确定将哪些外存作业调入内存之中，进而创建 PCB 等，插入就绪队列
- 一般用于批处理系统，分/实时系统一般直接入内存，无此环节

- 调度特性
  - 1. 接纳作业数（内存驻留数）
    - 太多 $\Rightarrow$ 周转时间 T 长
    - 太少 $\Rightarrow$ 系统效率低
  - 2. 接纳策略：
    - 即采用何种调度算法：FCFS、短作业优先、基于作业优先权的调度、响应比高者优先等

低级调度

- 主要是由分派程序（Dispatcher）分派处理机。
- 1. 非抢占方式:
  - 简单，实时性差 (如 win31)
- 2. 抢占方式
  - （1）时间片原则
  - （2）优先权原则
  - （3）短作业优先原则。

中级调度

- 为提高系统吞吐量和内存利用率而引入的内外存对换功能（换出时，进程为挂起或就绪驻外状态）
- 典型代表：存储器管理中的对换功能

选择调度方式和算法的准则 I

- 一、面向用户的准则: 周转时间、响应时间
  - 1. 周转时间短（常用于批处理系统）
    - 概念：作业从提交到完成的时间，分为：
      - （1）驻外等待调度时间
      - （2）驻内等待调度时间
      - （3）执行时间
      - （4）阻塞时间
  - 2. 响应时间快:（对交互型作业）
    - 概念：键盘提交请求到首次响应时间
      - （1）输入传送时间
      - （2）处理时间
      - （3）响应传送时间

选择调度方式和算法的准则 II

- 3. 截止时间的保证（特别于实时系统）
- 4. 优先权准则:（即需要抢占调度）

- 二、面向系统的准则
  - 1. 吞吐量高（特别于批处理）: 单位时间完成作业数
  - 2. 处理机利用率好:（因 CPU 贵，特别对于大中型多用户系统）
  - 3. 各类资源的平衡利用。(? 折算标准）

早期更看重处理机利用率（面向系统优先），后来更注重用户体验。

3.2 调度算法

- 3.2.1 先来先服务和短作业优先调度算法
- 3.2.2 高优先权优先调度算法
- 3.2.3 基于时间片的轮转调度算法

- OS 中调度的实质是一种资源分配

Scheduler and Dispatcher

- In simple term, Scheduler will select the process from ready queue and dispatcher will do the labor work of saving and loading of new process into the cpu.

  From: https://www.quora.com/OS-What-is-the-difference-between-a-scheduler-and-a-dispatcher

3.2.1 先来先服务和短作业优先

- 1.FCFS
  - 特点：简单，有利于长作业即 CPU 繁忙型作业
- 2. 短作业/进程优先调度算法：SJ(P)F
  - 提高了平均周转时间和平均带权周转时间（从而提高了系统吞吐量）
  - 特点：对长作业不利，有可能得不到服务（饥饿）
  - 估计时间不易确定

FCFS 示例

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

FCFS 示例

进程名	到达时间	服务时间	开始执行时间	完成时间	周转时间	带权周转时间
A	0	1	0	1	1	1
B	1	100	1	101	100	1
C	2	1	101	102	100	100
D	3	100	102	202	199	1.99

FCFS 和 SJF 的比较

	进程名	A	B	C	D	E	平均
	到达时间	0	1	2	3	4	
	服务时间	4	3	5	2	4	
FCFS	完成时间						
	周转时间						
	带权周转时间						
SJF	完成时间						
	周转时间						
	带权周转时间						

FCFS 和 SJF 的比较

	进程名	A	B	C	D	E	平均
	到达时间	0	1	2	3	4	
	服务时间	4	3	5	2	4	
FCFS	完成时间	4	7	12	14	18	
	周转时间	4	6	10	11	14	9
	带权周转时间	1	2	2	5.5	3.5	2.8
SJF	完成时间	4	9	18	6	13	
	周转时间	4	8	16	3	9	8
	带权周转时间	1	2.67	3.2	1.5	2.25	2.1

3.2.2 高优先权优先调度算法

- 优先权调度算法类型
  - 非抢占式优先权算法
  - 抢占式优先权算法，实时性更好。
- 优先权类型
  - 静态优先权
  - 动态优先权
- 高响应比优先

优先权类型 I

- 1. 静态优先权：
  - 进程优先权在整个运行期不变。
  - 确定优先权依据
    - （1）进程类型
    - （2）进程对资源的需求；
    - （3）根据用户需求。
  - 特点：简单，但低优先权作业可能长期不被调度。

优先权类型 II

- 2. 动态优先权:
  - 如: 优先权随执行时间而下降，随等待时间而升高。
  - 响应比 $R_p=$（等待时间 + 服务时间）/服务时间作为优先权
  - 优点: 长短兼顾
  - 缺点: 需计算 R_p

高响应比优先算法 I

- 特点
  - 响应比 $R_p = (tw + ts)/ts$
  - (1) 短作业 R_P 大。
  - (2) ts(要求服务时间) 相同的进程间相当于 FCFS。
  - (3) 长作业等待一段时间仍能得到服务。

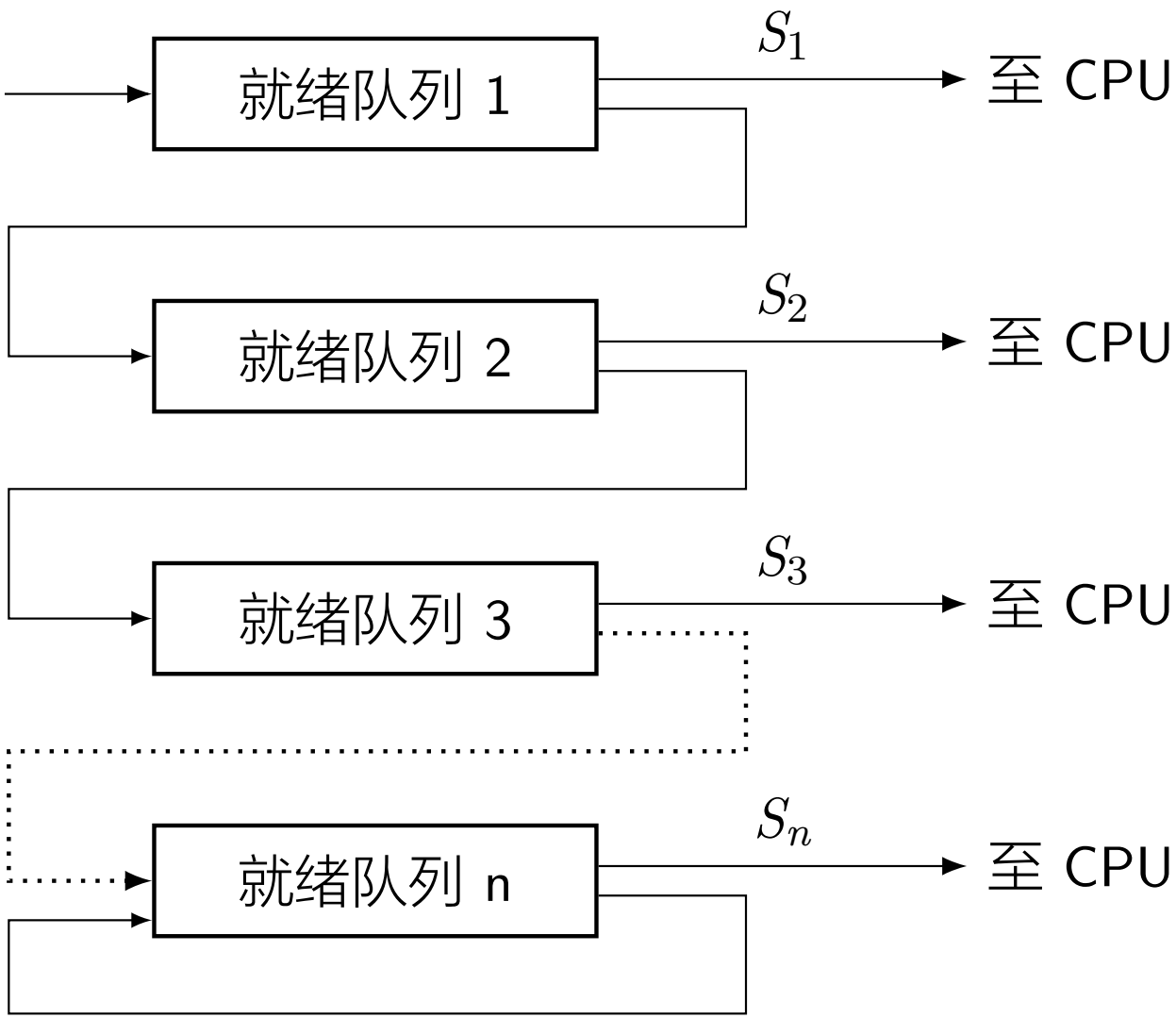
3.2.3 基于时间片的轮转调度算法 I

- 1. 时间片轮转
  - 时间片大小的确定
    - 太大：退化为 FCFS；
    - 太小：系统开销过大
  - 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：（应保证一个时间片处理完常用命令）

3.2.3 基于时间片的轮转调度算法 II

- 2. 多级反馈队列调度
  - 特点：长、短作业兼顾，有较好的响应时间
    - （1）短作业一次完成；
    - （2）中型作业周转时间不长；
    - （3）大型作业不会长期不处理。

多级队列反馈调度算法



时间片：$S_1 < S_2 < S_3$

3.3 实时调度

- 前面提到的调度算法，不能满足实时系统的要求

3.3.1 实现实时调度的基本条件 I

- 1. 提供必要的调度信息
  - (1) 就绪时间;
  - (2) 开始/完成截止时间;
  - (3) 处理时间;
  - (4) 资源要求;
  - (5) 优先级;
- 2. 系统处理能力强

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leqslant 1(\text{单处理器}) \quad or \quad \sum_{i=1}^{m} \frac{C_i}{P_i} \leqslant N(\text{多处理器})$$

  - C_i 为处理时间，P_i 为周期时间（基于周期性实时任务）

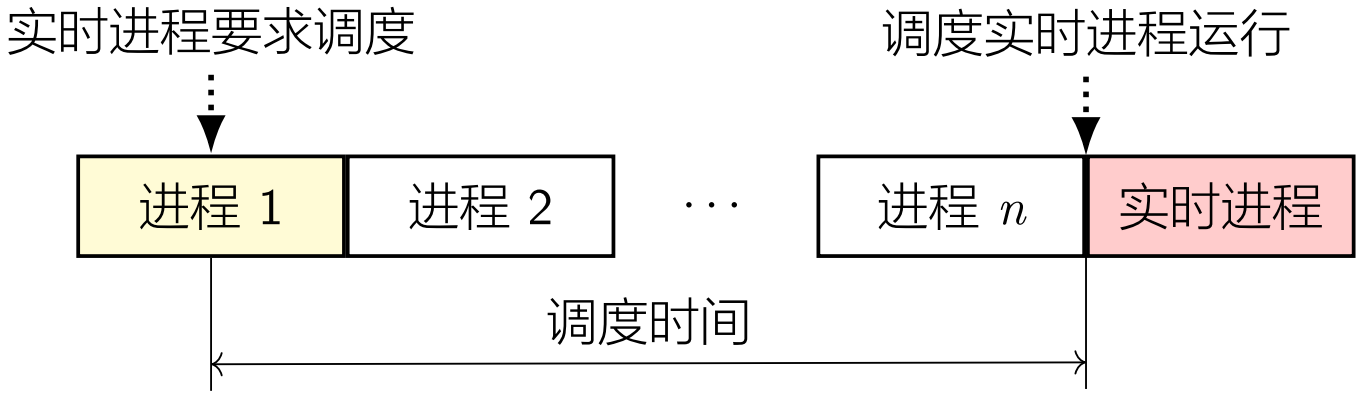
3.3.1 实现实时调度的基本条件 II

- 3. 采用抢占调度方式
  - 剥夺方式：一般都采用此
  - 非剥夺方式（实现简单）：一般应使实时任务较小，以及时放弃CPU。
- 4. 具有快速切换机制
  - 具有快速响应外部中断能力
  - 快速任务分派

3.3.2 实时调度算法的分类

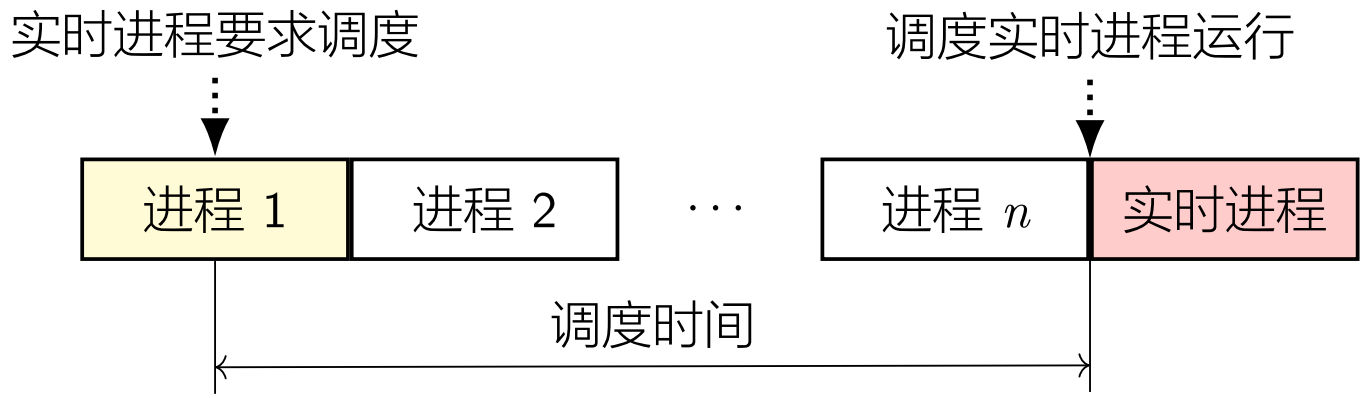
- 1. 非抢占式调度算法
  - 时间片轮转秒级
  - 非抢占优先权（协同）秒 毫秒级
- 2. 抢占式调度算法
  - 时钟中断抢占优先权毫秒级
    - 基于抢占点抢占
  - 立即抢占 immediate preemption 毫秒 微秒级
    - 只要不在临界区即抢占（中断引发）

调度时机图例

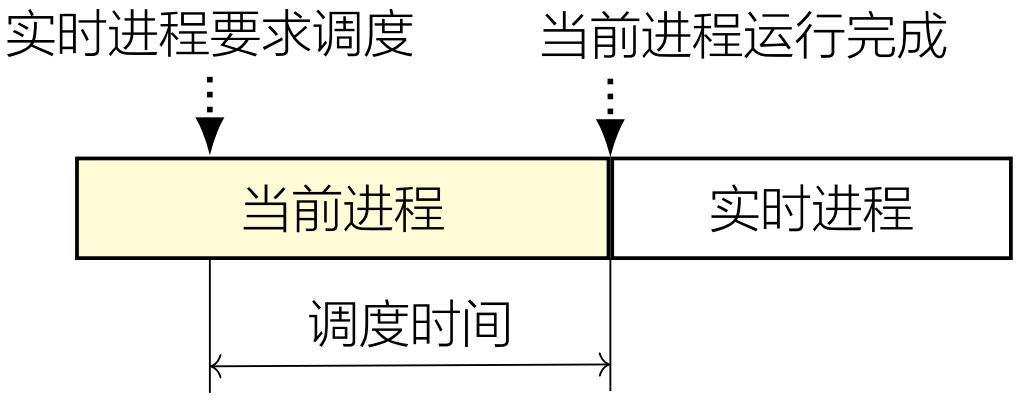


(a) 非抢占轮转调度

调度时机图例

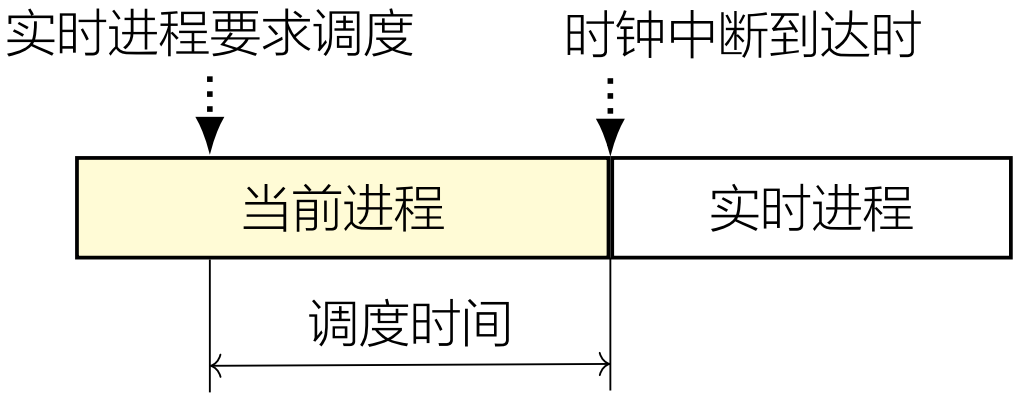


(a) 非抢占轮转调度



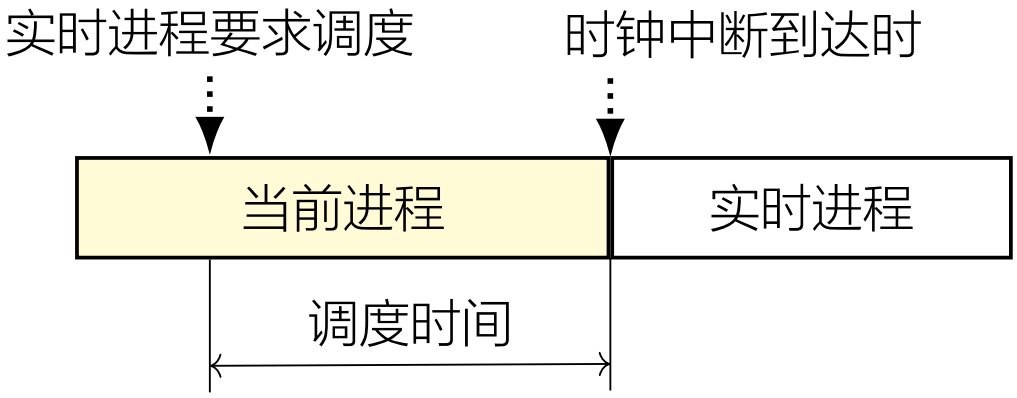
(b) 非抢占优先权调度

调度时机图例

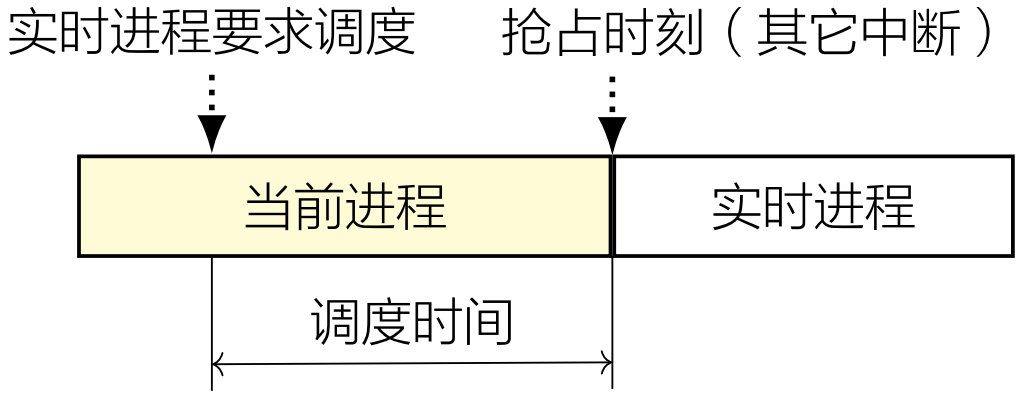


(c) 基于时钟中断抢占的优先权抢占调度

调度时机图例



(c) 基于时钟中断抢占的优先权抢占调度

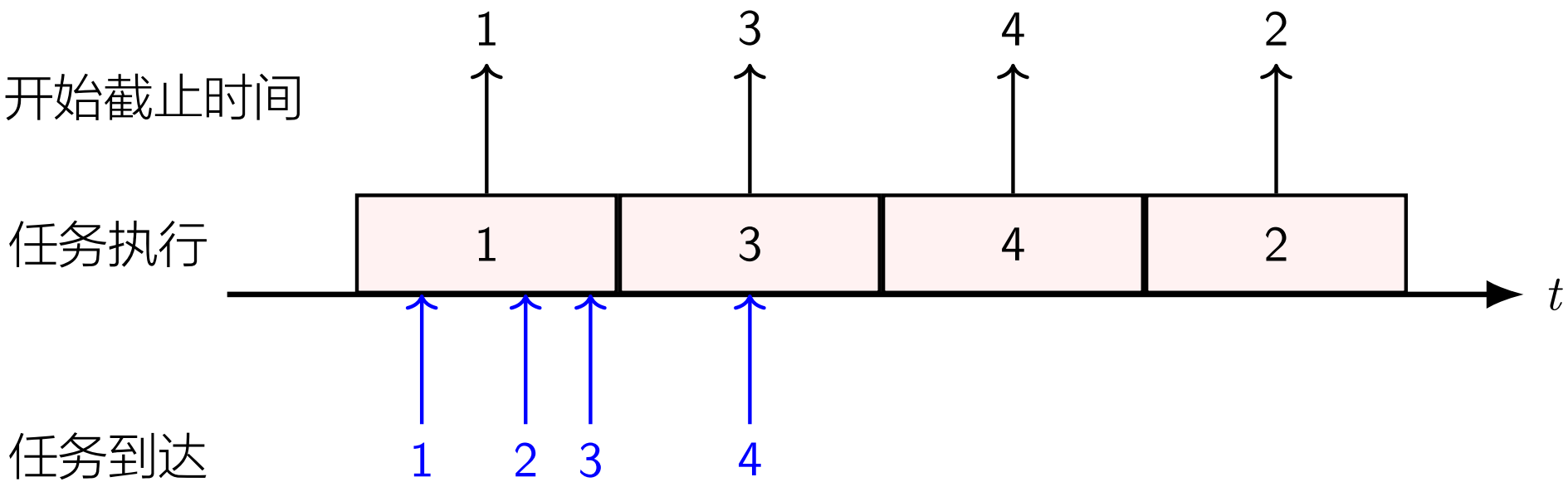


(d) 立即抢占优先权调度

3.3.3 常用的几种实时调度算法

- 1. 最早截止时间优先 EDF (earliest deadline first) 算法
  - 根据任务的截止时间来确定任务的优先级
  - 截止时间越早，优先级越高
  - 可以是抢占式或非抢占式

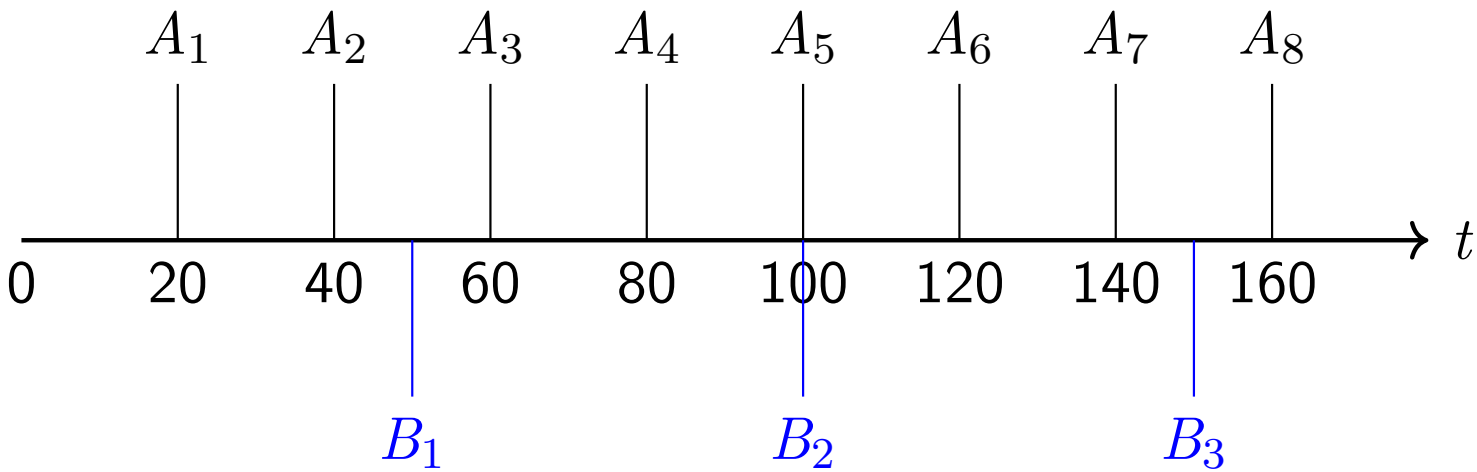
最早截止时间优先 EDF 例



EDF 算法用于非抢占调度方式

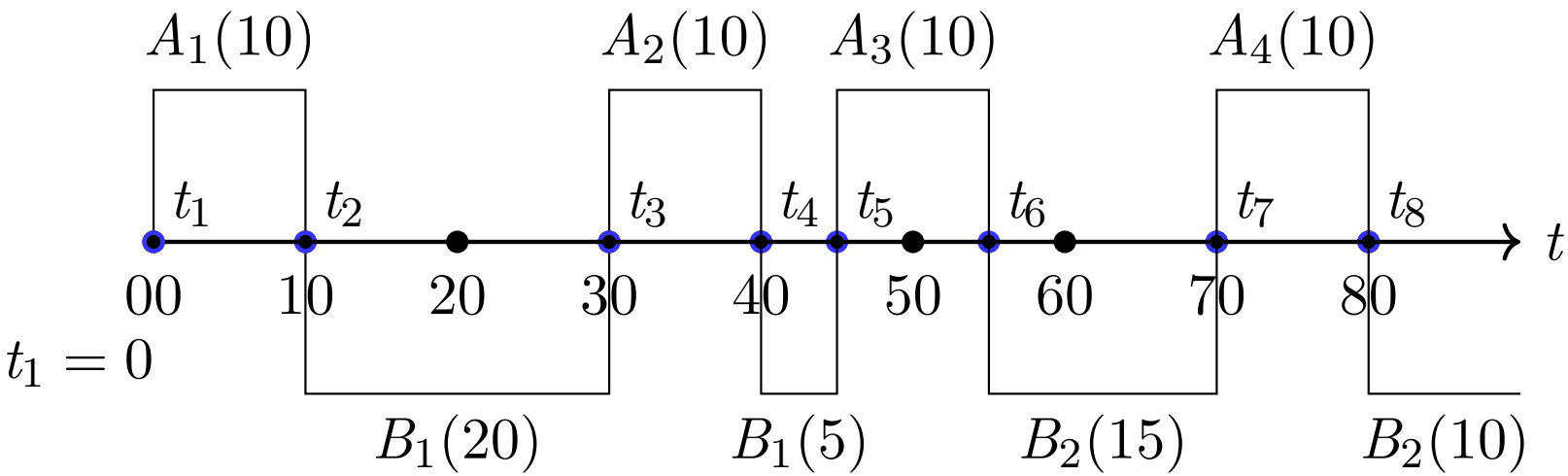
2 最低松弛度优先 LLF 算法

- 松弛度：
  - 若 A 进程需在 200ms 时完成，其本身运行需要 100ms，当前时刻是 10ms，则 A 的松弛度为：200 − 100 − 10 = 90
  - 主要用于可抢占的调度方式中
- 例：两个周期性实时任务 A、B，A 要求每 20ms 执行一次，每次需要 10ms，B 要求每 50ms 执行一次，每次需要 25 毫秒。



A/B 任务每次必须完成的时间

最低松弛度优先 LLF 算法 (2)



3.4 多处理机系统中的调度

- 提供计算机系统性能的主要途径
  - 提高元器件运行速度
  - 改进计算机体系结构

  - 20 世纪 70 年代出现多处理器系统 MPS
    - Multi-Processor Systems

3.4.1 多处理器系统的类型

MPS：MultiProcessor System

- 1. 紧密耦合 MPS 和松弛耦合 MPS
  - 紧密耦合
    - 共享 RAM 和 I/O
    - 高速总线和交叉开关连接
  - 松弛耦合
    - 独立 RAM 和 I/O
    - 通道和通信线路连接
- 2. 对称 MPS 和非对称 MPS
  - 对称多处理器系统 (SMPS: Symmetric MultiProcessor System)
  - 非对称多处理器系统

3.4.2 进程分配方式

- 1. 对称多处理器系统 (SMPS) 中进程分配方式
  - 静态分配
  - 动态分配
    - 可防止系统中多个处理器忙闲不均
- 2. 非 SMPS 中进程分配方式
  - 多采用主从方式 (Master – Slave)
  - 进程调度在主处理器上执行
  - 有潜在的不可靠性

3.4.3 进程 (线程) 调度方式

1. 自调度 (Self-Scheduling)
2. 成组调度方式 (Gang Scheduling)
3. 专用处理器分配方式

1. 自调度 (Self-Scheduling)

- 各个处理机自行在就绪队列中取任务。
- 特点；简单，分布式调度，调度算法可采用前述方法，多个 CPU 利用率都不错（不会闲）
- 但:
  - 瓶颈问题，（单队列）
  - 低效性；（需拷贝现场）
  - 线程切换频繁（当线程合作时，各线程并行的条件不容易满足）

2 成组调度方式 (Gang Scheduling)

- 将一个进程中的一组线程，每次分派时同时分派到一组处理机上执行，在剥夺处理机时也同时对这一组线程执行
- 优点：
  - (1) 对相互合作的进（线）程组调度，可以减小切换，减小系统开销。
  - (2) 每次分配一组 CPU，减少了调度频率。
- 成组调度如何为应用程序分配时间?
  - (1) 面向程序
  - (2) 面向线程：使处理机利用率更高。

2 成组调度方式 (Gang Scheduling)

	程序 A	程序 B
CPU_1	线程 1	线程 2
CPU_2	线程 2	空闲
CPU_3	线程 3	空闲
CPU_4	线程 4	空闲
时间	1/2	1/2

浪费 37.5%

	程序 A	程序 B
CPU_1	线程 1	线程 2
CPU_2	线程 2	空闲
CPU_3	线程 3	空闲
CPU_4	线程 4	空闲
时间	4/5	1/5

浪费 15%

- 面向应用程序：每个程序各获得 50% 执行时间
- 面向线程：两个程序共有 5 个线程，第一个程序获得 4/5，第二个获得 1/5

3 专用处理器分配方式

- 为进程中的每个线程都固定分配一个 CPU，直到该线程执行完毕
- 引入：
  - 多处理机系统，每个处理机已不再属宝贵资源。
- 特点：
  - 每个进（线）程专用处理机，使其切换小，提高效率。
- 主要用于大型计算，实时系统

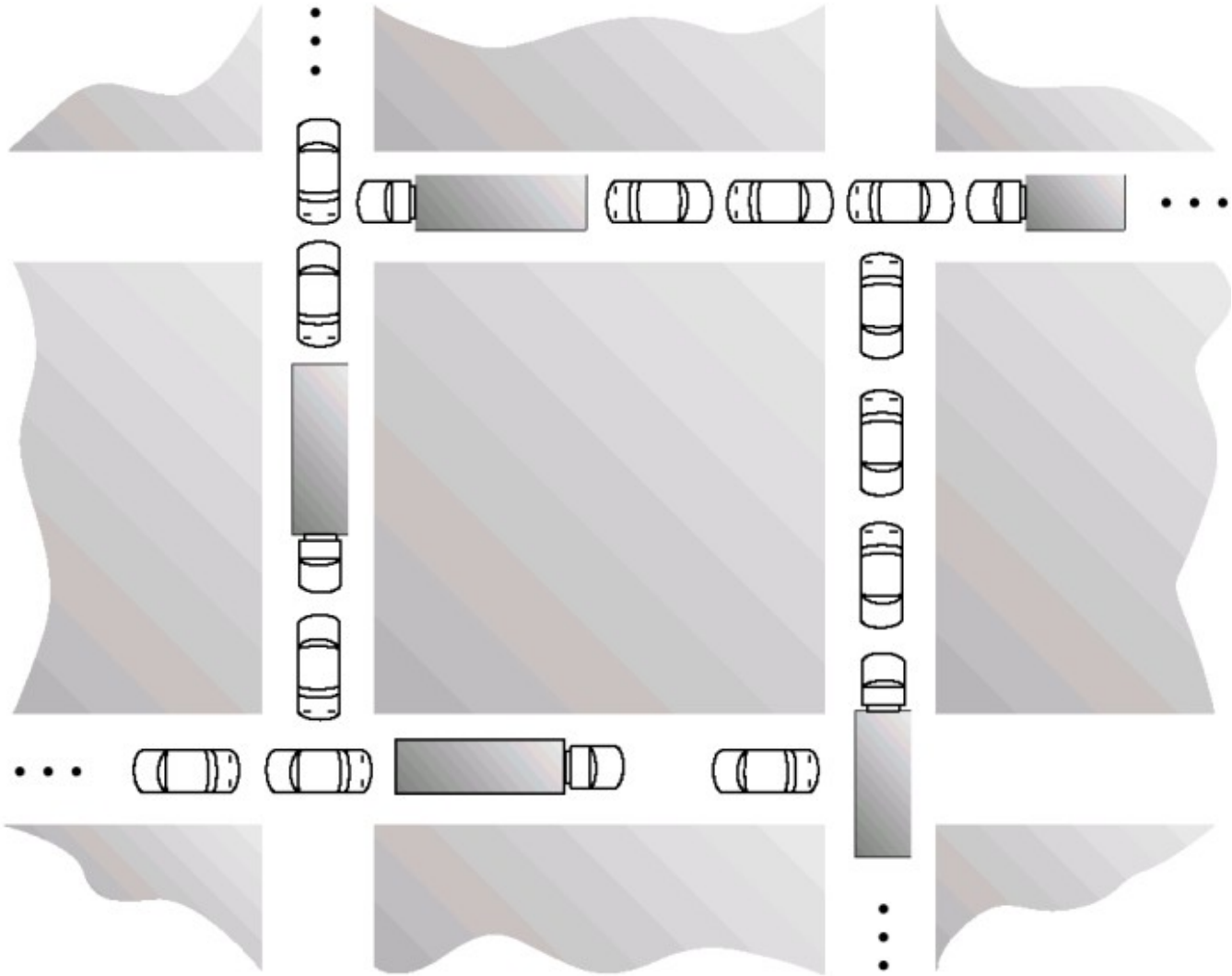
Next

3.5 产生死锁的原因和必要条件

- 并发可以改善系统的资源利用率，但可能发生死锁 (Deadlock)
- 死锁：
  - 多个进程在运行过程中因争夺资源而造成的一种僵局 (Deadly-Embrace)，当进程处于这种僵持状态时，若无外力作用，它们都将无法再向前推进。

死锁的产生

- Deadlock in the everyday life



若干死锁的例子

- 例 1 进程推进顺序不当产生死锁
  - 设系统有打印机、读卡机各一台，被进程P和Q共享。两个进程并发执行，按下列次序请求和释放资源：

进程 P
请求读卡机
请求打印机
释放读卡机
释放打印机

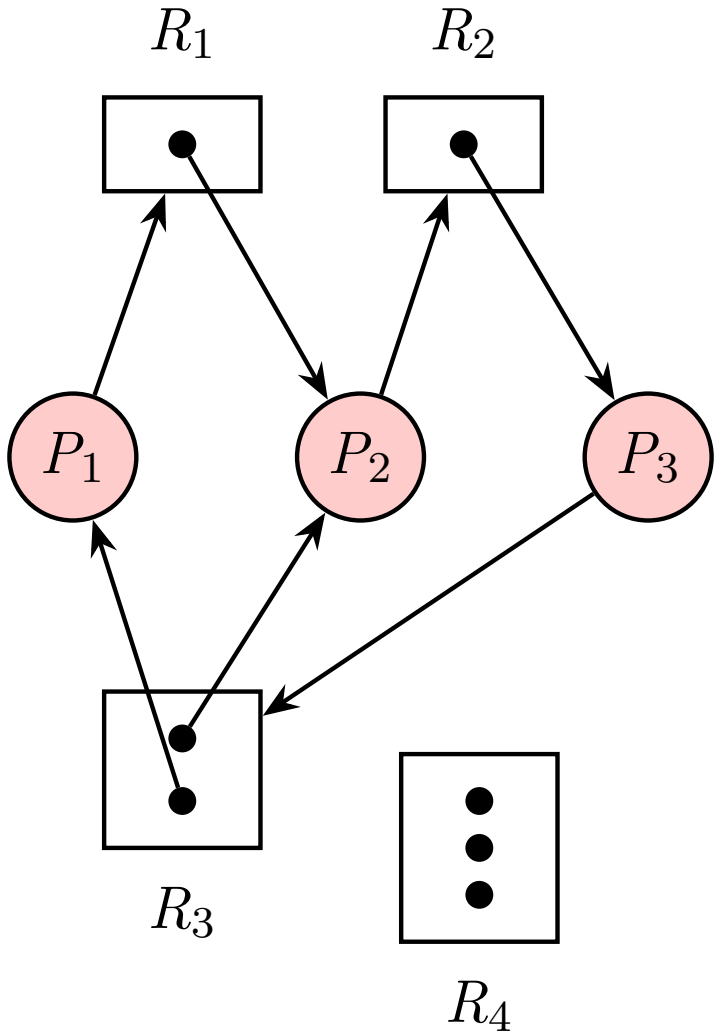
进程 Q
请求打印机
请求读卡机
释放读卡机
释放打印机

例 2 PV 操作使用不当产生死锁

进程 P
$P(S_1)$;
$P(S_2)$;
使用资源 r_1 和 r_2;
$V(S_1)$;
$V(S_2)$;

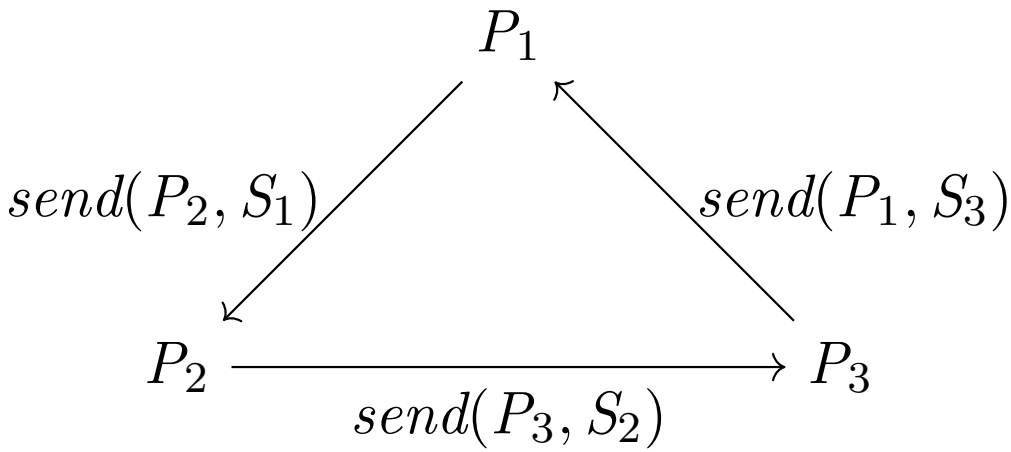
进程 P
$P(S_2)$;
$P(S_1)$;
使用资源 r_1 和 r_2;
$V(S_2)$;
$V(S_1)$;

例 3 资源分配不当引起死锁



例 4 对临时性资源使用不加限制引起死锁

- 进程通信使用的信件是一种临时性资源，如果对信件的发送和接收不加限制，可能引起死锁。



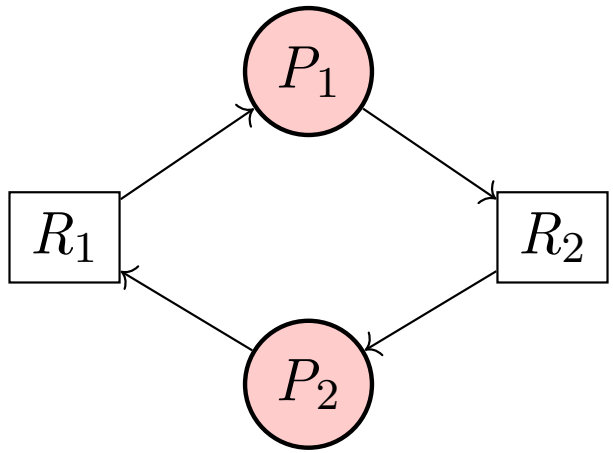
上图中，进程 P_1 需要接收到由 P_3 发送来的资源 S_3 后，就可以向进程 P_2 发送资源 S_1 ...

3.5.1 产生死锁的原因

1. 竞争资源引起死锁
2. 进程推进顺序不当引起死锁

1. 竞争资源引起死锁

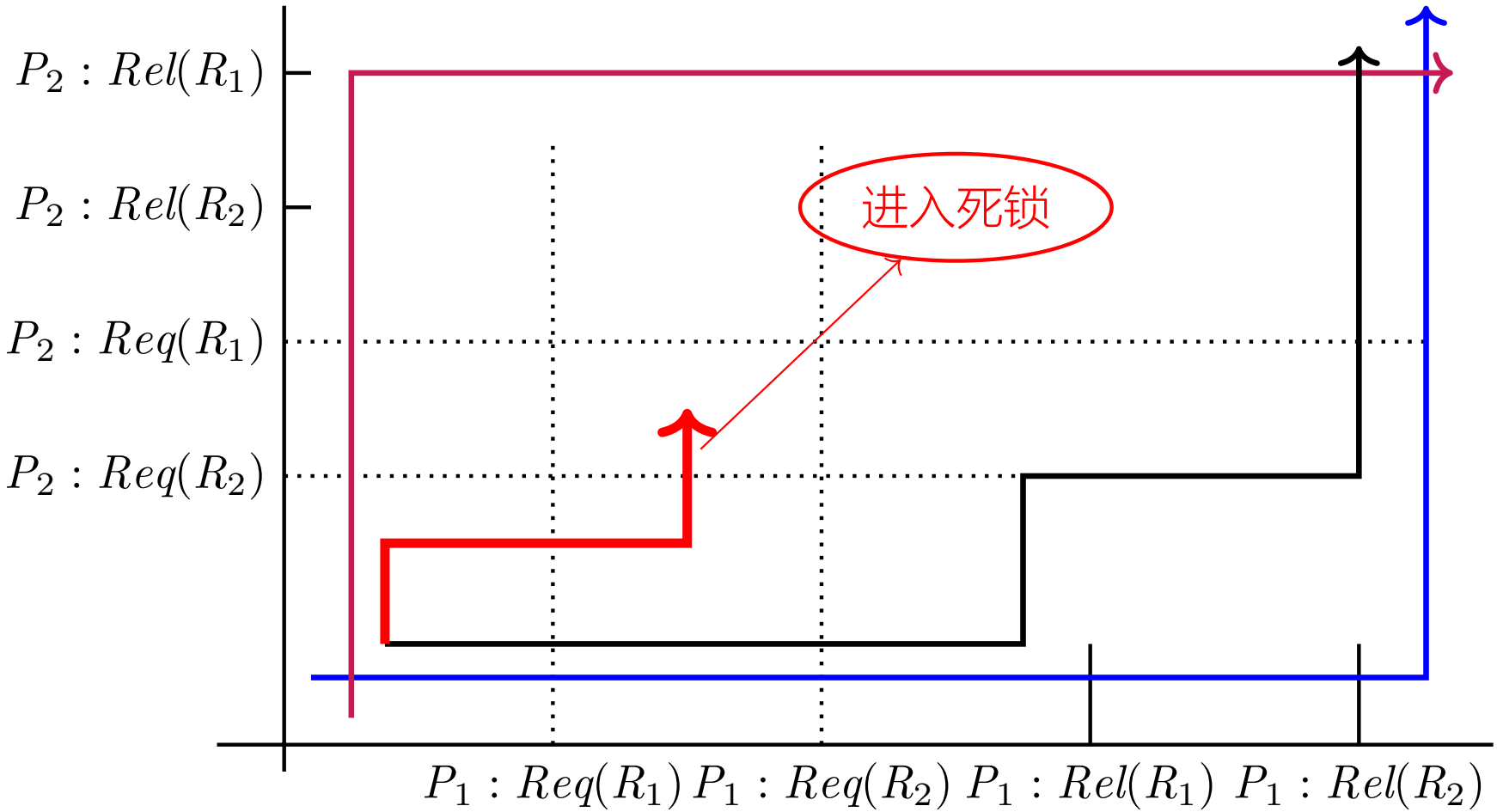
- 1. 可剥夺（CPU、内存，）和非剥夺性（打印机，磁带机）资源
- 2. 竞争非剥夺性资源——可造成死锁



- 3. 竞争临时性资源
  - 永久性资源可重复使用，临时性资源是指由一个进程产生，被另一个进程使用一段时间后便无用的资源，也称为消耗性资源。

2. 进程推进顺序不当引起死锁

- 下图四种颜色的线表示了四种不同的推进顺序，其中红色线所代表的推进顺序会导致死锁



3.5.2 产生死锁的必要条件

- 1. 互斥条件（Mutual Exclusion）
  - 共享资源不会导致死锁，如只读文件
  - 某些资源，如打印机、磁带等，需要被单个进程互斥访问
- 2. 请求和保持条件 (Hold and Wait)
  - 持有等待
- 3. 不剥夺条件 (No Preemption)
  - 不能抢占
- 4. 环路等待 (Circular Wait)
  - 循环等待

3.5.3 处理死锁的基本方法

- 不允许死锁发生
  - 1. 静态预防;
    - 破坏 4 个条件之一：有效，使资源利用率低。
  - 2. 动态避免
    - 防止进入不安全态。
- 允许死锁发生
  - 3. 顺其自然、不予理睬
  - 4. 检测 + 解除：检测到死锁再清除。

3.6 死锁预防和避免

- 死锁预防
  - 破坏四个条件之一
- 死锁避免
  - 通过银行家算法判断系统安全状态

3.6.1 死锁预防

- 一、互斥条件是资源固有属性，不能避免。
- 二、摒弃请求和保持条件
  - 全分配，全释放（AND）
  - 缺点：(1) 延迟进程运行; (2) 资源严重浪费
- 三、摒弃“不剥夺”条件
  - 增加系统开销，且进程前段工作可能失效。
- 四、摒弃“环路”条件
  - 有序资源分配法：为资源编号，申请时需按编号进行。
  - 缺点：(1) 新增资源不便 (原序号已排定); (2) 用户不自由; (3) 资源与进程使用顺序不同造成浪费

3.6.2 死锁避免之系统安全状态

- 按某种顺序并发进程都能达到获得最大资源而顺序完成的序列为安全序列。
- 能找到安全序列的状态为安全状态。

安全状态举例

进程	最大需求	已分配	可用
P_1	10	5	3
P_2	4	2	
P_3	9	2	

安全序列：$P_2 \rightarrow P_1 \rightarrow P_3$

安全状态向不安全状态的转换

- 上例中，若 P3 再申请一台，则不安全

进程	最大需求	已分配	可用
P_1	10	5	2
P_2	4	2	
P_3	9	3	

3.6.3 利用银行家算法避免死锁

- 1. 数据结构
  - $available[j] = k$: 系统现有 R_j 类资源 k 个;
  - $max[i, j] = k$: 进程 i 需要 R_j 的最大数 k 个;
  - $alloc[i, j] = k$: 进程 i 已得到 R_j 类资源 k 个;
  - $need[i, j] = k$: 进程 i 需要 R_j 类资源 k 个
    - $\therefore need[i, j] = max[i, j] - alloc[i, j]$
  - $request_i$: 进程 i 请求资源数
  - $work_i$: 进程 i 执行完后系统应有资源数（也即可用数）
  - $finish[i]$: 布尔量，表进程 i 能否顺序完成。

示例

	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
										(2	3	0)
P_1	3	2	2	2	0	0	1	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_0 时刻的资源分配表

示例

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

T_0 时刻的安全序列

示例

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

P_1 申请资源 (1,0,2) 时安全性检查 (安全)

示例

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

为 P_0 分配 $(0, 2, 0)$ 后的情况 (不安全)

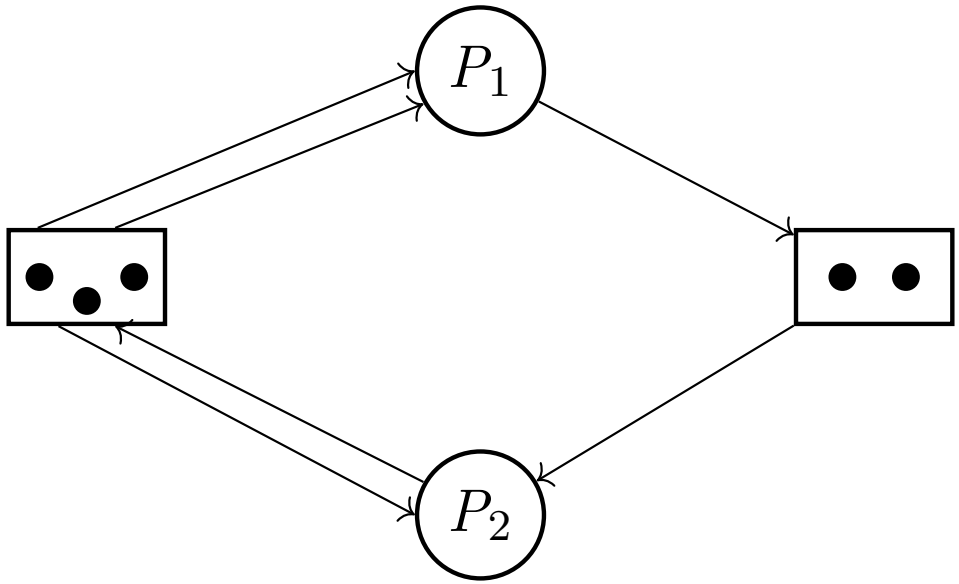
3.7 死锁的检测与解除

- 3.7.1 死锁的检测

3.7.1 死锁的检测

- 1. 资源分配图
- 2. 死锁定理
- 3. 检测死锁的算法

1. 资源分配图



2. 死锁定理

- 简化资源分配图
- 若能完全简化则消去所有的边
- 定理：
  - 死锁状态的充分条件，资源分配图不可完全简化

3. 检测死锁的算法

Algorithm 1 死锁检测算法

1: work = available;
2: $L = \{L_i | alloc_i = 0, req_i = 0\}$ //孤立进程点
3: **for** all $L_i \in L$ **do**
4: **for** all $req_i \leq work$ **do**
5: $work = work + alloc_i$
6: $L = L_i \cup L$
7: Deadlock $= \neg(L = \{p_1, \cdots, p_n\})$

3.7.2 死锁的解除

- 检测到死锁后，回退到上一状态（要进行资源剥夺，且需保存以前状态的分配信息），重新分配，若不行，继续回退……
- 策略
  - 抢占资源
  - 杀死进程
  - 上翻：rollback
  - 难点：
    - 难以准确判断是否有死锁
    - 计算量大
    - 问题：检查死锁本身出问题

讨论：如何应对死锁?

- 检测修复与动态避免成本较高
  - 实现的程序复杂性
  - 运行时间
- 综合治理
  - 对 CPU、内存实施可抢占的静态防止策略 (破坏不剥夺条件)
  - 对磁盘、打印机实施假脱机共享
  - 一些特殊资源按顺序访问
  - 剩余的其他问题不予理睬

课堂作业

- 设系统中有 3 种类型的资源（A，B，C）和 5 个进程 P1、P2、P3、P4、P5。A 资源的数量为 17,B 资源的数量为 5,C 资源的数量为 20。在 T_0 时刻系统状态见下表所示。系统采用银行家算法实施死锁避免策略。
- T_0 时刻系统状态

	最大资源需求量			已分配资源数量		
	A	B	C	A	B	C
P_1	5	5	9	2	1	2
P_2	5	3	6	4	0	2
P_3	4	0	11	4	0	5
P_4	4	2	5	2	0	4
P_5	4	2	4	3	1	4

问题

- (1) T_0 时刻是否为安全状态？若是，请给出安全序列。
- (2) 在 T_0 时刻若进程 P_2 请求资源 $(0, 3, 4)$，是否能实施资源分配？为什么？
- (3) 在 (2) 的基础上，若进程 P_4 请求资源 $(2, 0, 1)$ 是否能实施资源分配？为什么？
- (4) 在 (3) 的基础上，若进程 P_1 请求资源 $(0, 2, 0)$ 是否能实施资源分配？为什么？

复习题 I

1. 如果系统中有 n 个进程，则在等待队列中进程的个数最多可为 ( ) 个
2. 在操作系统中，不可中断执行的操作称为 ( )
3. 在有 m 个进程的系统中出现死锁时，死锁进程的个数 k 应该满足的条件是 ( )
4. 不让死锁发生的策略可以分为静态和动态两种，死锁避免属于 ( )
5. 若使当前运行进程总是优先级最高的进程，则应该选择进程 ( ) 调度算法
6. 在操作系统中引起进程调度的因素有：( )、( )、( )、( )、( )

复习题 II

7. 产生进程死锁的根本原因是（　　），另一个基本原因是（　　）.
8. 进程是一个程序对某个数据集的（　　）
9. 在操作系统中引入线程的主要目的是（　　）
10. 进程调度算法采用等时间片轮转法时，时间片过大，就会使轮转法转化为（　　）调度算法
11. 并发进程中涉及到相同变量的程序段叫做（　　），对这些程序段要（　　）执行。
12. 死锁产生的四个必要条件是：互斥控制、(请求和保持)、（　　）、（　　）。
13. 某程序运行时经常要打印中间结果。计算时，该进程处于（　　）态，打印时处于（　　）态，打印结束时进程处于（　　）态。

复习题 III

14. 操作系统中，可以并行工作的基本单位是（　　），它是由程序、（　　）和（　　）组成。
15. 进程初建时处于（　　）态，运行时因为时钟中断而处于（　　）态，因等事件或资源而处于（　　）态
16. 当处理机空闲时，进程调度程序从（　　）队列中选取一个进程执行。
17. 当采用静态资源分配法预防死锁时，破坏了产生死锁的四个必要条件中的（　　）条件。

本章调查作业

1. CPU 与 GPU
2. 开源软件简史 [1]

[1] https://linuxstory.org/simple-history-about-opensource-1/

END