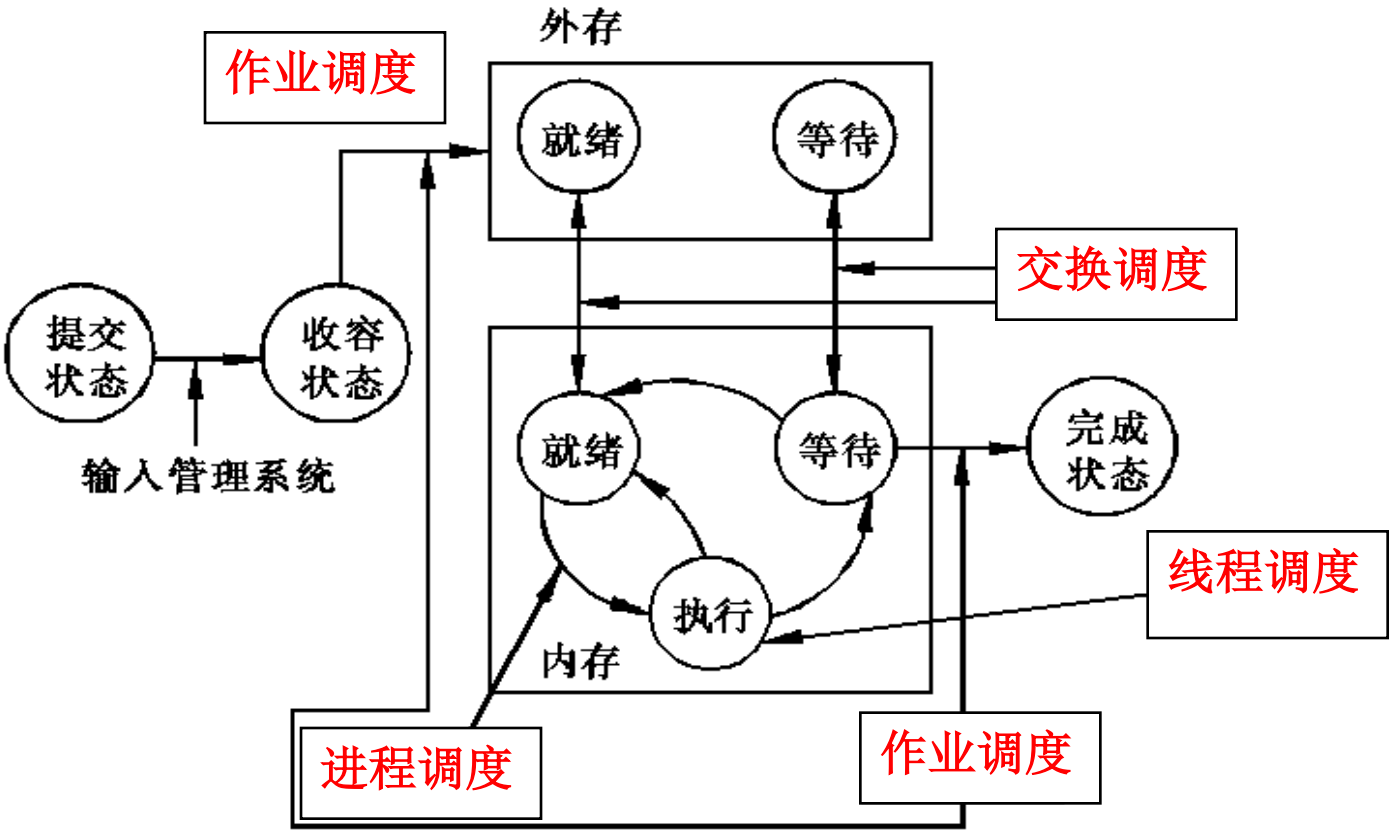
操作系统

第三章 处理机调度与死锁

第三章 处理机调度与死锁

补充：作业的状态及其转换

一个作业从提交给计算机系统到执行结束退出系统，一般都要经历**提交、收容、执行和完成**四个状态。



3.1 处理机调度的基本概念

3.1.1 高级、中级和低级调度

1. 高级调度(High Scheduling)，即作业调度

在每次执行作业调度时，须做出以下两个决定

1) 接纳多少个作业

2) 接纳哪些作业

调度目标：

(1) 对所有作业应该是公平合理的；
(2) 应使设备有高的利用率；
(3) 单位时间内执行尽可能多的作业；
(4) 有快的响应时间。

2. 低级调度(Low Level Scheduling)，又称进程调度

原因：

进程数>处理机数，导致进程互相争夺处理机。

这就要求进程调度程序按一定的策略，动态地把处理机分配给处于**就绪队列**中的某一个进程，以使之执行。

2. 低级调度(Low Level Scheduling)

1)非抢占方式(Non-preemptive Mode)

可能引起进程调度的因素可归结为这样几个：

① 正在执行的进程执行完毕， 或因发生某事件而不能再继续执行；

② 执行中的进程因提出I/O请求而暂停执行；

③ 在进程通信或同步过程中执行了某种原语操作，如P操作(wait操作)、Block原语、Wakeup原语等。

非抢占调度方式的优点是实现简单、系统开销小，适用于大多数的批处理系统环境。但它难以满足紧急任务的要求——立即执行，因而可能造成难以预料的后果。

显然，在要求比较严格的实时系统中，不宜采用这种方式。

2) 抢占方式(Preemptive Mode)

抢占的原则有：

- **优先级原则：**

  高优先级抢夺低优先级；

- **短进程优先原则：**

  短进程抢夺长进程；

- **时间片原则：**

  时间片完重新调度；

- **强制剥夺：**

  极重要的进程，人工干预，强制调度。

进程调度的方式：

1剥夺方式(抢占方式)：

用于实时和分时系统

2非剥夺方式（非抢占方式）：

只有进程在阻塞或执行完毕后，才产生调度。

3选择剥夺方式：

给每个进程设置特征位0，1。

特征位为1的进程可以剥夺特征位为0的进程处理机的使用权。

特征位相同的按非剥夺方式。

引起进程调度的原因有：（重新分配CPU的时机）

(1) 进程正常执行完毕。

(2) 正在执行的进程自己调用阻塞原语将自己阻塞起来进入等待状态。

(3) 正在执行的进程调用了P原语，因资源不足而被阻塞；或调用了V原语激活了等待资源队列中的高优先级的进程(剥夺方式)。

(4)进程提出I/O请求后被阻塞(让权等待)。

(5) 在分时系统中时间片已经用完。

(6)处理机从管态（系统态）返回目态（用户态）时，可调度新的用户进程执行，或者相反。

关于进程调度的补充内容

主要介绍：

① 记录系统中所有进程的执行情况；
② 选择占有处理机的进程的策略；
③ 进程调度引起的进程上下文切换等。

（1）记录系统中所有进程的执行情况：

- 将各进程的执行情况和当前状态特征记录在各进程的PCB表中。
- 将各进程的PCB表（根据状态特征和资源需求）排成相应的队列并进行动态队列转接及管理。
- 在处理机空闲的时侯。分配给就绪队列中的一个进程。

(2) 选择占有处理机的进程的策略：

按照一定的策略选择一个处于就绪状态的进程，使其获得处理机执行。这些策略决定了调度算法的性能。

对于不同的系统设计目的，有不同的选择策略。

如：系统开销较少的静态优先数调度法；

适合于分时系统的轮转法和多级反馈轮转法等。

(3) 进行进程上下文切换

进程上下文(context):

包括进程的状态、变量和数据结构的值、硬件寄存器的值和PCB以及有关程序等。

进程的执行是在其上下文中执行。当正在执行的进程由于某种原因要让出处理机时，系统要做进程上下文切换，以使另一个进程得以执行。

当进行上下文切换时，系统要:

- 首先检查是否允许做上下文切换(优先级);
- 保留被切换进程的当前信息(现场)，以便返回该进程时，从断点恢复该进程的执行。

系统保留了CPU现场之后，**调度程序**选择一个新的处于就绪状态的进程，并装配(布置)该进程的上下文(现场)，使CPU的控制权转换到被选中进程中。

进程上下文切换

进程上下文由正文段、数据段、硬件寄存器的内容以及有关数据结构等组成。

硬件寄存器主要包括存放CPU将要执行的下条指令虚拟地址的程序计数器PC。

指出机器与进程相关联的硬件状态的处理机状态寄存器PS。

存放过程调用（或系统调用）时所传递参数的通用寄存器R以及堆栈指针寄存器S等。

数据结构则包括PCB等在内的所有与执行该进程有关的管理和控制用表格、数组、链表等。

在发生进程调度时，系统要做进程上下文切换。

进程上下文切换包括4个步骤：

(1) 决定是否做上下文切换以及是否允许做上下文切换。包括对进程调度原因的检查分析，以及当前执行进程的资格和CPU执行方式的检查等。

(2) 保存当前执行进程的上下文（现场，断点）。

(3) 选择一个处于就绪状态的新进程。

(4) 装配（布置）所选新进程的上下文（恢复现场，断点），将CPU使用权交给所选进程。

进程调度性能评价

是系统的低级调度，调度的优劣直接影响作业调度的性能。
衡量方法可分为**定性**和**定量**两种。

定性衡量：

调度的可靠性（切换时，不能破坏数据结构）；

简洁性（不能繁琐复杂）

定量评价：

CPU的利用率评价；

进程在就绪队列中的等待时间与执行时间之比等。

3. 中级调度(Medium-Term Scheduling)，即交换调度

主要目的：

腾出足够的内存空间，提高内存利用率和系统吞吐量。

- 按照给定的策略，将处于外存交换区中的就绪状态或等待状态的进程调入内存；
- 或把处于内存中就绪状态或等待状态的进程交换到外存交换区。

详见第二章节的挂起suspend() 和激活active()

3.1.2 调度队列模型

1.仅有进程调度的调度队列模型

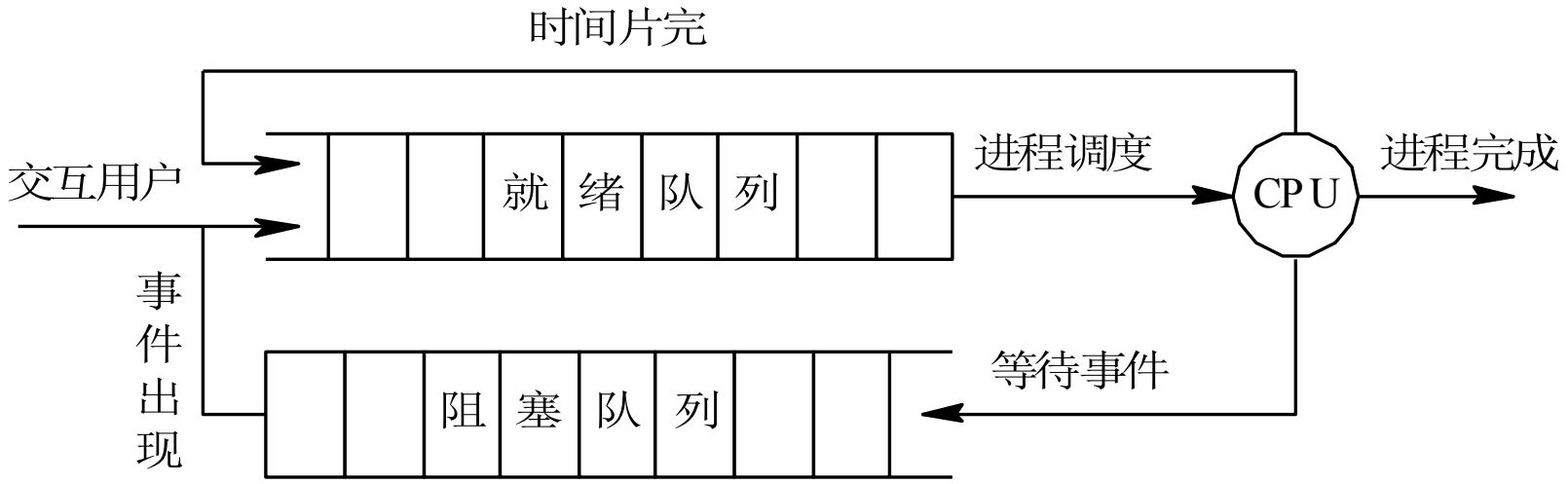


图 3 - 1 仅具有进程调度的调度队列模型

2. 具有高级和低级调度的调度队列模型

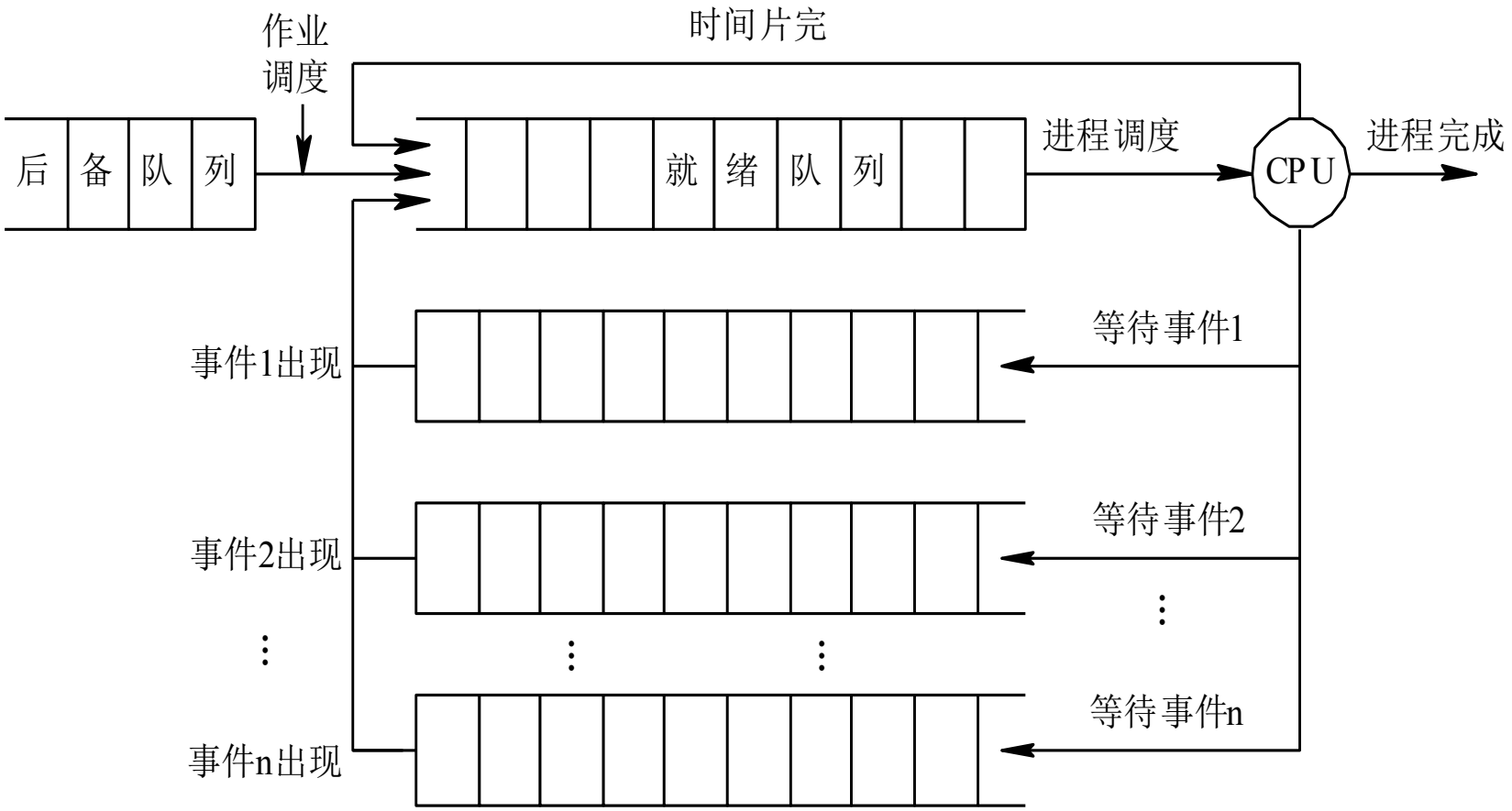


图 3-2 具有高、低两级调度的调度队列模型

3. 同时具有三级调度的调度队列模型

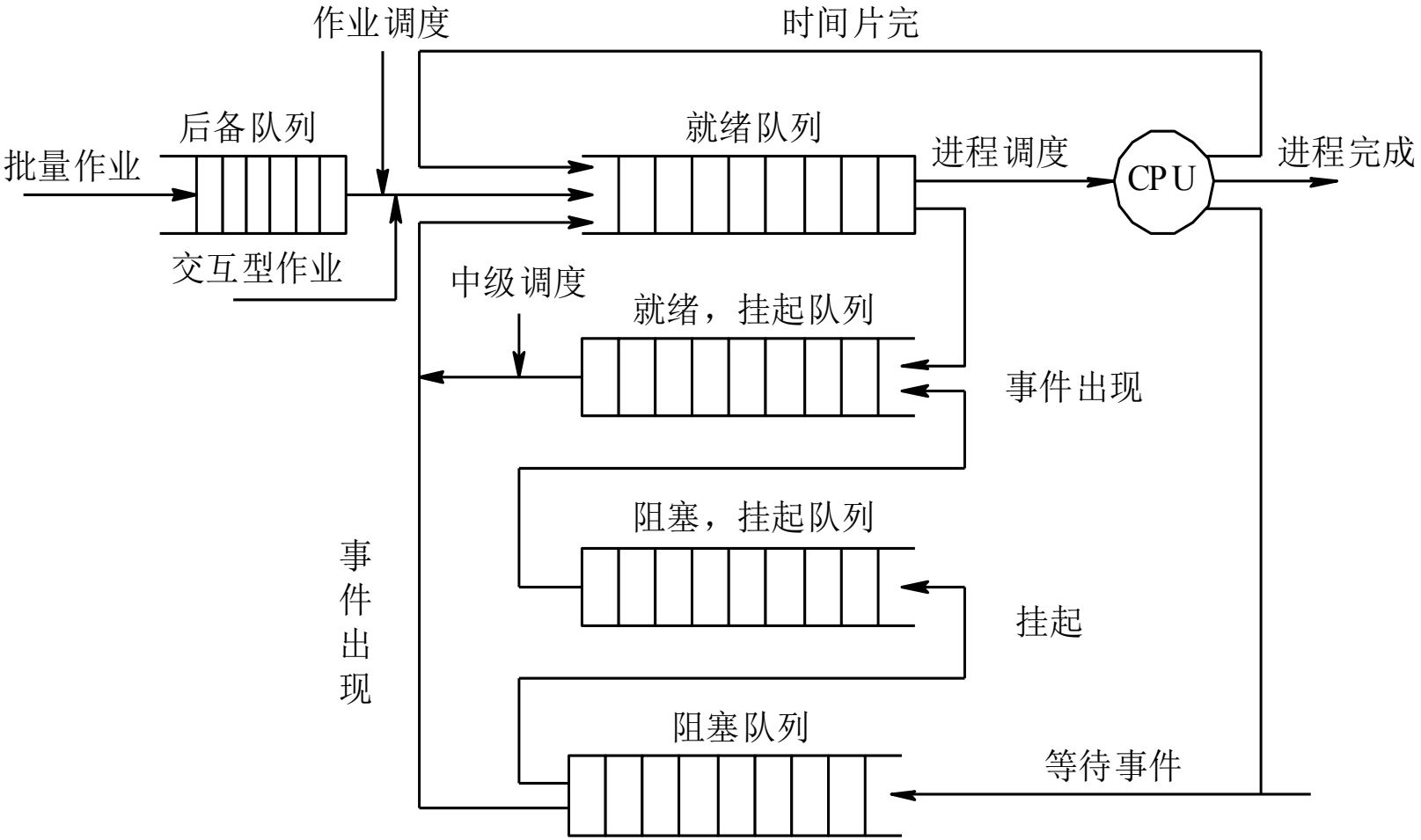


图 3-3 具有三级调度时的调度队列模型

3.1.3 选择调度方式和调度算法的若干准则

1. 面向用户的准则

(1)周转时间短。

(2) 响应时间快。

(3) 截止时间的保证。

(4) 优先权准则。

大多数操作系统都根据用户需要，采用兼顾某些目标的简单调度算法。

批处理系统：

用作业的平均周转时间或平均带权周转时间，被作为衡量调度算法优劣的标准。

分时系统和实时系统：

用作业的平均响应时间被作为衡量调度策略优劣的标准。

1. 周转时间：

作业i的周转时间T_i为

$$T_i=T_{ei}-T_{si}$$ 完成时间 - 提交时间

其中T_{ei}为作业i的完成时间，T_{si}为作业的提交时间。

对于被测定作业流所含有的n（n>=1）个作业来说，其平均周转时间为：

$$T=\frac{1}{n}\sum_{i=1}^{n}T_i$$

一个作业的周转时间也是该作业在系统内停留的时间。包含两部分：

$$T_i=T_{wi}+T_{ri}$$ 等待时间+执行时间

这里，T_{wi}主要指作业i由后备状态到执行状态的等待时间，它不包括作业进入执行状态后的等待时间。

2. 带权周转时间

为了更进一步反映调度性能，使用带权周转时间的概念。

$W_i=T_i/T_r$ 作业周转时间/作业执行时间：

对于被测定作业流所含有的几个作业来说，其平均带权周转时间为：

$$W=\frac{1}{n}\sum_{i=1}^{n}W_i$$

而对于分时系统，除了要保证系统吞吐量大、资源利用率高之外，还应保证有用户能够容忍的响应时间。

2. 面向系统的准则

(1)系统吞吐量高。

(2) 处理机利用率好。

(3) 各类资源的平衡利用。

第三章 处理机调度与死锁

3.2 调 度 算 法

3.2.1 先来先服务和短作业(进程)优先调度算法

1. 先来先服务调度算法

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

3.2.1 先来先服务和短作业(进程)优先调度算法

调度算法 \ 作业情况	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 (a)	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 (b)	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	3.1	1.5	2.25	2.1

图 3-4 FCFS和SJF调度算法的性能

2. 短作业(进程)优先调度算法

短作业(进程)优先调度算法SJ(P)F，是指对短作业或短进程优先调度的算法。它们可以分别用于作业调度和进程调度。

- **短作业优先(SJF)调度算法**

  从后备队列中选择一个或若干个估计运行时间最短的作业，将它们调入内存运行。

- **短进程优先(SPF)调度算法**

  从**就绪队列**中选出一估计运行时间最短的进程，将处理机分配给它，使它立即执行并一直执行到完成，或发生某事件而被阻塞放弃处理机时，再重新调度。

3.2.2 高优先权优先调度算法

1. 优先权调度算法的类型

1）非抢占式优先权算法

系统一旦把处理机分配给就绪队列中优先权最高的进程后，该进程便一直执行下去，直至完成； 或因发生某事件使该进程放弃处理机时，系统方可再将处理机重新分配给另一优先权最高的进程。

这种调度算法主要用于批处理系统中；

也可用于某些对实时性要求不严的实时系统中。

2）抢占式优先权调度算法

系统把处理机分配给优先权最高的进程，使之执行。

但在其执行期间，只要又出现了另一个其优先权更高的进程，进程调度程序就立即停止当前进程(原优先权最高的进程)的执行，重新将处理机分配给新到的优先权最高的进程。

每当系统中出现一个新的就绪进程i时，就将其优先权P_i与正在执行的进程j的优先权P_j进行比较:

如果$P_i \leq P_j$，原进程P_j便继续执行；

如果$P_i > P_j$, 则做进程切换,使Pi进程投入执行.

2. 优先权的类型

1) 静态优先权

静态优先权是在创建进程时确定的，且在进程的整个运行期间保持不变。

2) 动态优先权

动态优先权是指，在创建进程时所赋予的优先权，是可以随进程的推进或随其等待时间的增加而改变的，以便获得更好的调度性能。

3. 高响应比优先调度算法

优先权的变化规律可描述为：

$$\text{优先权} = \frac{\text{等待时间} + \text{服务时间}}{\text{服务时间}}$$

(1) 如果作业的等待时间相同，则要求服务的时间愈短，其优先权愈高，因而该算法有利于短作业。

(2) 当要求服务的时间相同时，作业的优先权决定于其等待时间，等待时间愈长，其优先权愈高，因而它实现的是先来先服务。

(3) 对于长作业，作业的优先级可以随等待时间的增加而提高，当其等待时间足够长时，其优先级便可升到很高，从而也可获得处理机。

3.2.3 基于时间片的轮转调度算法

1.时间片轮转法

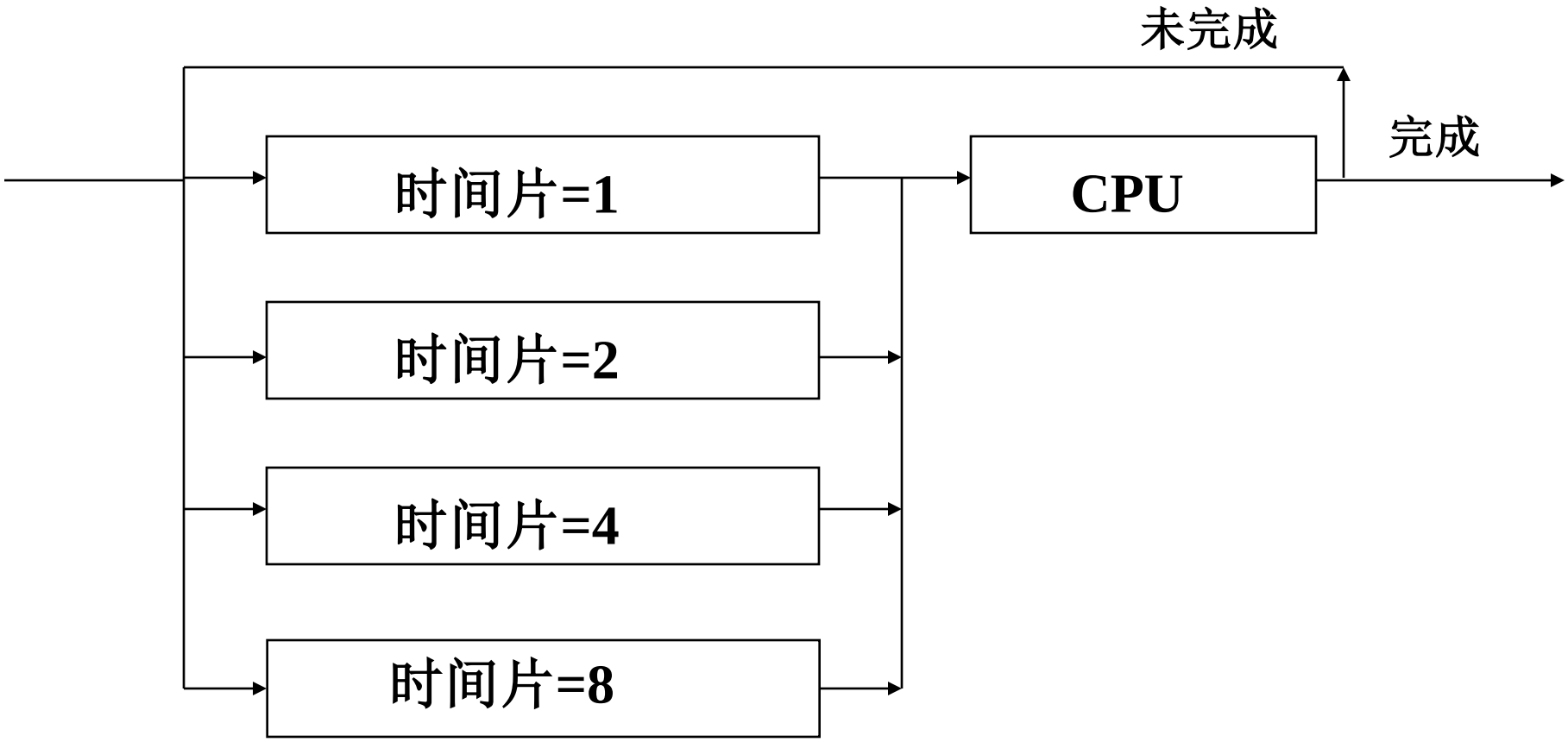
系统将所有的就绪进程按先来先服务的原则，排成一个队列，每次调度时，把CPU分配给队首进程，并令其执行一个时间片。时间片的大小从几ms到几百ms。当执行的时间片用完时，由一个计时器发出时钟中断请求，调度程序便据此信号来停止该进程的执行，并将它送往就绪队列的末尾；然后，再把处理机分配给就绪队列中新的队首进程，同时也让它执行一个时间片。这样就可以保证就绪队列中的所有进程，在一给定的时间内，均能获得一时间片的处理机执行时间。

2. 多级反馈队列调度算法

多个就绪队列，各个队列赋予不同的优先级。 第一个队列的优先级最高，第二个队列次之，其余各队列的优先权逐个降低。

各个队列中进程执行时间片的大小也各不相同，在优先权愈高的队列中，为每个进程所规定的执行时间片就愈小。例如，第二个队列的时间片要比第一个队列的时间片长一倍，……，第*i*+1个队列的时间片要比第i个队列的时间片长一倍。

多级反馈轮转法示意图



1) 在规定的时间片内若未完成任务，进下一级队列；

2) 当且仅当上一级队列无进程时，才调度下一级队列的进程执行。

3. 多级反馈队列调度算法的性能

(1) 终端型作业用户。

(2) 短批处理作业用户。

(3) 长批处理作业用户。

第三章 处理机调度与死锁

3.3 实 时 调 度

3.3.1 实现实时调度的基本条件

1. 提供必要的信息

(1)就绪时间。

(2) 开始截止时间和完成截止时间。

(3) 处理时间。

(4) 资源要求。

(5) 优先级。

2. 系统处理能力强

在实时系统中，通常都有着多个实时任务。若处理机的处理能力不够强，则有可能因处理机忙不过来而使某些实时任务不能得到及时处理，从而导致发生难以预料的后果。假定系统中有m个周期性的硬实时任务，它们的处理时间可表示为C_i，周期时间表示为P_i，则在单处理机情况下，必须满足下面的限制条件，系统才是可调度的。

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

2. 系统处理能力强

假如系统中有6个硬实时任务，它们的周期时间都是 50 ms，而每次的处理时间为 10 ms，则不难算出，此时是不能满足上式的，因而系统是不可调度的。

解决的方法是提高系统的处理能力，其途径有二：其一仍是采用单处理机系统， 但须增强其处理能力， 以显著地减少对每一个任务的处理时间；其二是采用多处理机系统。假定系统中的处理机数为N，则应将上述的限制条件改为：

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq N$$

3. 采用抢占式调度机制

当一个优先权更高的任务到达时，允许将当前任务暂时挂起，而令高优先权任务立即投入运行，可满足该硬实时任务对截止时间的要求。但这种调度机制比较复杂。

对于一些小的实时系统，如果能预知任务的开始截止时间，则对实时任务的调度可采用非抢占调度机制，以简化任务调度所花费的系统开销。但在设计这种调度机制时，应使所有的实时任务都比较小，并在执行完关键性程序和临界区后，能及时地将自己阻塞起来，以便释放出处理机， 供调度程序去调度那种开始截止时间即将到达的任务。

4. 具有快速切换机制

该机制应具有如下两方面的能力：

(1) 对外部中断的快速响应能力。为使在紧迫的外部事件请求中断时系统能及时响应，要求系统具有快速硬件中断机构，还应使禁止中断的时间间隔尽量短， 以免耽误时机(其它紧迫任务)。

(2) 快速的任务分派能力。在完成任务调度后，便应进行任务切换。为了提高分派程序进行任务切换时的速度， 应使系统中的每个运行功能单位适当的小，以减少任务切换的时间开销。

3.3.2 实时调度算法的分类

1. 非抢占式调度算法

(1) 非抢占式轮转调度算法。

(2) 非抢占式优先调度算法。

2. 抢占式调度算法

(1) 基于时钟中断的抢占式优先权调度算法。
(2) 立即抢占(Immediate Preemption)的优先权调度算法。

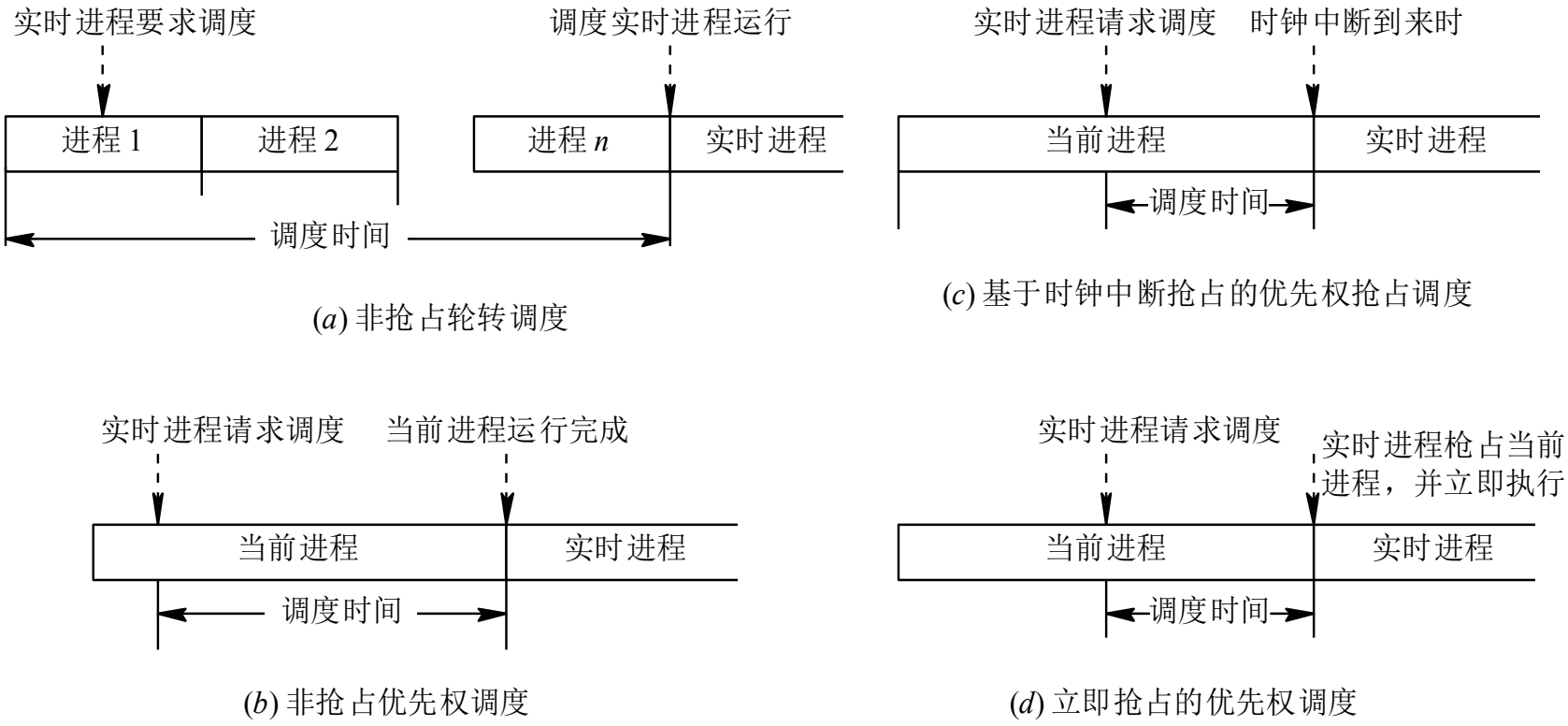


图 3-6 实时进程调度

3.3.3 常用的几种实时调度算法

1. 最早截止时间优先算法

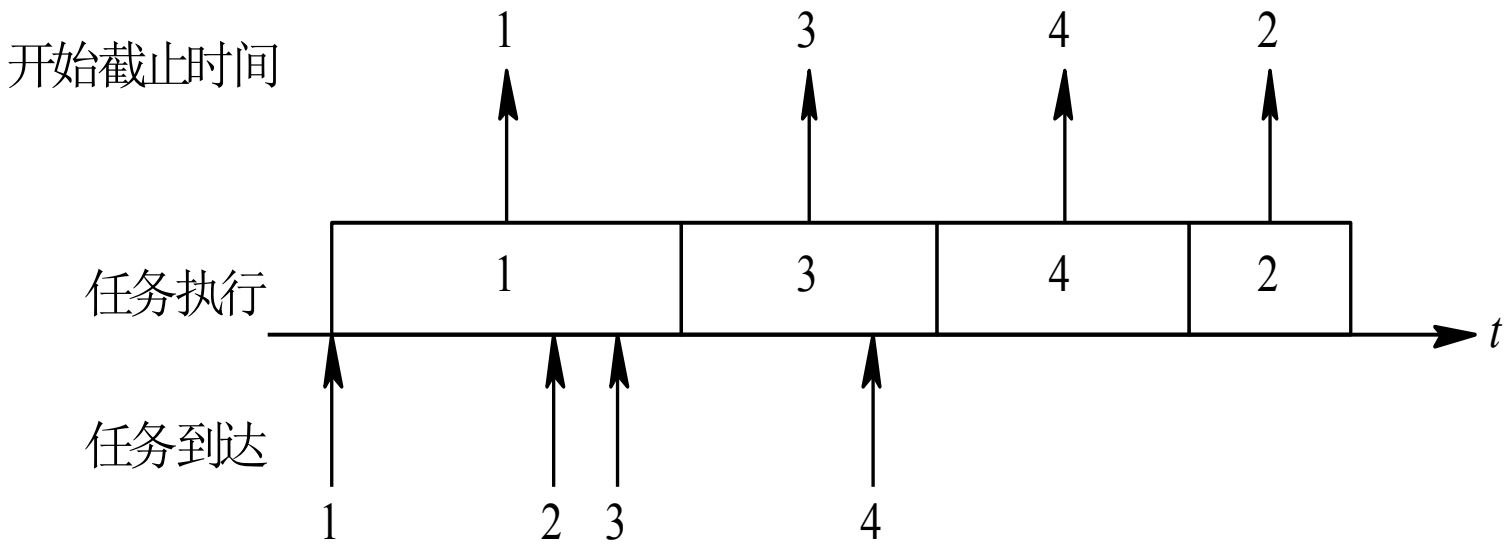


图 3-7 EDF算法用于非抢占调度方式

2. 最低松弛度优先即LLF(Least Laxity First)算法

该算法是根据任务紧急(或松弛)的程度，来确定任务的优先级。任务的紧急程度愈高，为该任务所赋予的优先级就愈高， 以使之优先执行。例如，一个任务在200ms时必须完成，而它本身所需的运行时间就有100ms，因此，调度程序必须在100 ms之前调度执行，该任务的紧急程度(松弛程度)为100 ms。在实现该算法时要求系统中有一个按松弛度排序的实时任务就绪队列，松弛度最低的任务排在队列最前面，调度程序总是选择就绪队列中的队首任务执行。该算法主要用于可抢占调度方式中。

第三章 处理机调度与死锁

3.4 多处理机系统中的调度

3.4.1 多处理器系统的类型

(1) 紧密耦合(Tightly Coupted)MPS。

这通常是通过高速总线或高速交叉开关，来实现多个处理器之间的互连的。它们共享主存储器系统和I/O设备，并要求将主存储器划分为若干个能独立访问的存储器模块，以便多个处理机能同时对主存进行访问。系统中的所有资源和进程，都由操作系统实施统一的控制和管理。

3.4.1 多处理器系统的类型

(2) 松散耦合(Loosely Coupled)MPS。

在松散耦合MPS中，通常是通过通道或通信线路，来实现多台计算机之间的互连。每台计算机都有自己的存储器和I/O设备，并配置了OS来管理本地资源和在本地运行的进程。因此，每一台计算机都能独立地工作， 必要时可通过通信线路与其它计算机交换信息，以及协调它们之间的工作。

2. 对称多处理器系统和非对称多处理器系统

(1) 对称多处理器系统

在系统中所包含的各处理器单元，在功能和结构上都是相同的，当前的绝大多数MPS都属于SMP系统。例如，IBM公司的SR/6000 Model F50, 便是利用4片Power PC处理器构成的。

(2) 非对称多处理器系统

在系统中有多种类型的处理单元， 它们的功能和结构各不相同，其中只有一个主处理器，有多个从处理器。

3.4.2 进程分配方式

1. 对称多处理器系统中的进程分配方式

在SMP系统中，所有的处理器都是相同的，因而可把所有的处理器作为一个处理器池(Processor pool)，由调度程序或基于处理器的请求， 将任何一个进程分配给池中的任何一个处理器去处理。在进行进程分配时，可采用以下两种方式之一。

1) 静态分配(Static Assigenment)方式

2) 动态分配(Dynamic Assgement)方式

2. 非对称MPS中的进程分配方式

对于非对称MPS， 其OS大多采用主–从(Master-Slave)式OS，即OS的核心部分驻留在一台主机上(Master)， 而从机(Slave)上只是用户程序， 进程调度只由主机执行。 每当从机空闲时， 便向主机发送一索求进程的信号， 然后， 便等待主机为它分配进程。 在主机中保持有一个就绪队列， 只要就绪队列不空， 主机便从其队首摘下一进程分配给请求的从机。从机接收到分配的进程后便运行该进程， 该进程结束后从机又向主机发出请求。

3.4.3 进程(线程)调度方式

1. 自调度(Self-Scheduling)方式

1) 自调度机制囍

在多处理器系统中，自调度方式是最简单的一种调度方式。它是直接由单处理机环境下的调度方式演变而来的。 在系统中设置有一个公共的进程或线程就绪队列， 所有的处理器在空闲时，都可自己到该队列中取得一进程(或线程)来运行。在自调度方式中，可采用在单处理机环境下所用的调度算法，如先来先服务(FCFS)调度算法、最高优先权优先(FPF)调度算法和抢占式最高优先权优先调度算法等。

1. 自调度(Self-Scheduling)方式

2) 自调度方式的优点

首先，系统中的公共就绪队列可按照单处理机系统中所采用的各种方式加以组织，其调度算法也可沿用单处理机系统所用的算法，

其次， 只要系统中有任务，或者说只要公共就绪队列不空，就不会出现处理机空闲的情况，也不会发生处理器忙闲不均的现象，因提高处理器的利用率。

3) 自调度方式的缺点：

(1)瓶颈问题 (2) 低效性 (3) 线程切换频繁。

2. 成组调度(Gang Scheduling)方式

在成组调度时，如何为应用程序分配处理器时间，

1) 面向所有应用程序平均分配处理器时间

2) 面向所有线程平均分配处理器时间

	应用程序 A	应用程序 B
处理器 1	线程 1	线程 1
处理器 2	线程 2	空闲
处理器 3	线程 3	空闲
处理器 4	线程 4	空闲
	1/2	1/2

(a) 浪费 37.5%

	应用程序 A	应用程序 B
处理器 1	线程 1	线程 1
处理器 2	线程 2	空闲
处理器 3	线程 3	空闲
处理器 4	线程 4	空闲
	4/5	1/5

(b) 浪费 15%

第三章 处理机调度与死锁

3.5 产生死锁的原因和必要条件

3.5.1 产生死锁的原因

(1) 竞争资源。

1) 可剥夺和非剥夺性资源
2) 竞争非剥夺性资源
3) 竞争临时性资源

(2) 进程间推进顺序非法。

3.5.2 产生死锁的必要条件

(1)互斥条件

(2) 请求和保持条件

(3) 不剥夺条件

(4) 环路等待条件

3.5.3 处理死锁的基本方法

(1)预防死锁。

(2) 避免死锁。

(3) 检测死锁。

(4) 解除死锁。

第三章 处理机调度与死锁

3.6 预防死锁的方法

3.6.1 预防死锁

1. 摒弃“请求和保持”条件

2. 摒弃“不剥夺”条件

3. 摒弃“环路等待”条件

3.6.2 系统安全状态

1. 安全状态

在避免死锁的方法中，允许进程动态地申请资源，但系统在进行资源分配之前，应先计算此次资源分配的安全性。若此次分配不会导致系统进入不安全状态，则将资源分配给进程； 否则，令进程等待。

所谓安全状态，是指系统能按某种进程顺序(P_1，P_2，…，P_n)(称〈P_1, P_2, …, P_n〉序列为安全序列)，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，使每个进程都可顺利地完成。如果系统无法找到这样一个安全序列，则称系统处于不安全状态。

2. 安全状态之例

我们通过一个例子来说明安全性。假定系统中有三个进程P_1、 P_2和P_3，共有12台磁带机。进程P_1总共要求10台磁带机，P_2和P_3分别要求4台和9台。假设在T_0时刻，进程P_1、P_2和P_3已分别获得5台、2台和2台磁带机，尚有3台空闲未分配，如下表所示：

进 程	最 大 需 求	已 分 配	可 用
P_1	10	5	
P_2	4	2	3
P_3	9	2	

3. 由安全状态向不安全状态的转换

如果不按照安全序列分配资源，则系统可能会由安全状态进入不安全状态。例如，在T_0时刻以后，P_3又请求1台磁带机，若此时系统把剩余3台中的1台分配给P_3，则系统便进入不安全状态。 因为，此时也无法再找到一个安全序列， 例如，把其余的2台分配给P_2，这样，在P_2完成后只能释放出4台，既不能满足P_1尚需5台的要求，也不能满足P_3尚需6台的要求，致使它们都无法推进到完成，彼此都在等待对方释放资源，即陷入僵局，结果导致死锁。

3.6.3 利用银行家算法避免死锁

1. 银行家算法中的数据结构

(1) 可利用资源向量Available。这是一个含有*m*个元素的数组，其中的每一个元素代表一类可利用的资源数目，其初始值是系统中所配置的该类全部可用资源的数目，其数值随该类资源的分配和回收而动态地改变。如果Available［j］=K，则表示系统中现有R_j类资源*K*个。

(2) 最大需求矩阵Max。这是一个*n*×*m*的矩阵，它定义了系统中*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i需要R_j类资源的最大数目为K。

3.6.3 利用银行家算法避免死锁

(3) 分配矩阵Allocation。这也是一个$n \times m$的矩阵，它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。如果Allocation［i,j］=K，则表示进程i当前已分得R_j类资源的数目为K。囂

(4) 需求矩阵Need。这也是一个$n \times m$的矩阵，用以表示每一个进程尚需的各类资源数。如果Need［i,j］=K，则表示进程i还需要R_j类资源K个，方能完成其任务。

Need［i,j］=Max［i,j］-Allocation［i,j］

2. 银行家算法

设$Request_i$是进程P_i的请求向量，如果$Request_i$［j］=*K*，表示进程P_i需要*K*个R_j类型的资源。当P_i发出资源请求后，系统按下述步骤进行检查：囂

(1) 如果$Request_i$［j］≤Need［i,j］，便转向步骤2；否则认为出错，因为它所需要的资源数已超过它所宣布的最大值。

(2) 如果$Request_i$［j］≤Available［j］，便转向步骤(3)；否则， 表示尚无足够资源，P_i须等待。

2. 银行家算法

(3) 系统试探着把资源分配给进程P_i，并修改下面数据结构中的数值：

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］;

(4) 系统执行安全性算法，检查此次资源分配后，系统是否处于安全状态。若安全，才正式将资源分配给进程P_i，以完成本次分配；否则， 将本次的试探分配作废，恢复原来的资源分配状态，让进程P_i等待。

3. 安全性算法

(1) 设置两个向量：

① 工作向量Work: 它表示系统可提供给进程继续运行所需的各类资源数目，它含有*m*个元素，在执行安全算法开始时，Work：=Available;

② Finish: 它表示系统是否有足够的资源分配给进程，使之运行完成。开始时先做Finish［i］：=false; 当有足够资源分配给进程时， 再令Finish［i］：=true。

(2) 从进程集合中找到一个能满足下述条件的进程：

① Finish［i］=false;

② Need［i,j］≤Work［j］；

若找到， 执行步骤(3)， 否则，执行步骤(4)。

3. 安全性算法

(3) 当进程P_i获得资源后，可顺利执行，直至完成，并释放出分配给它的资源，故应执行：

Work［j］：=Work［j］+Allocation［i,j］;

Finish［i］：=true;

go to step 2;

(4) 如果所有进程的Finish［i］=true都满足， 则表示系统处于安全状态；否则，系统处于不安全状态。

第三章 处理机调度与死锁

3.7 死锁的检测与解除

3.7.1 死锁的检测

1.资源分配图(Resource Allocation Graph)

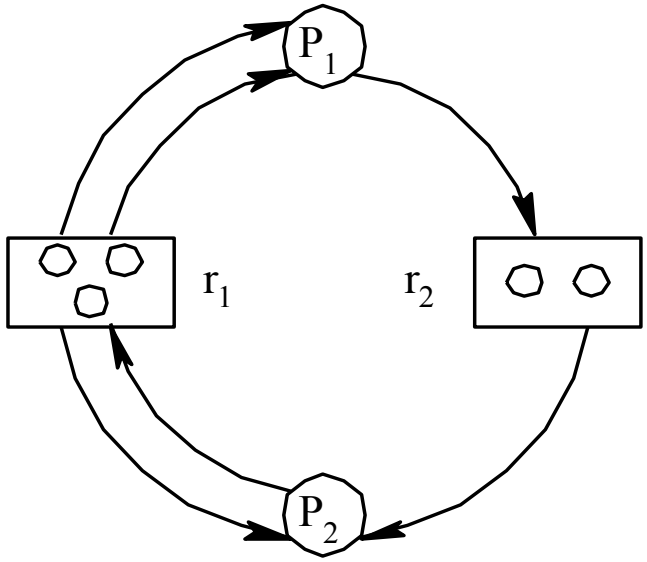


图 3-9 每类资源有多个时的情况

(2) 凡属于E中的一个边e∈E，都连接着P中的一个结点和R中的一个结点，

e={p_i, r_j}是资源请求边，由进程p_i指向资源r_j， 它表示进程p_i请求一个单位的r_j资源。

e={r_j, p_i}是资源分配边，由资源r_j指向进程p_i, 它表示把一个单位的资源r_j分配给进程p_i。

2. 死锁定理

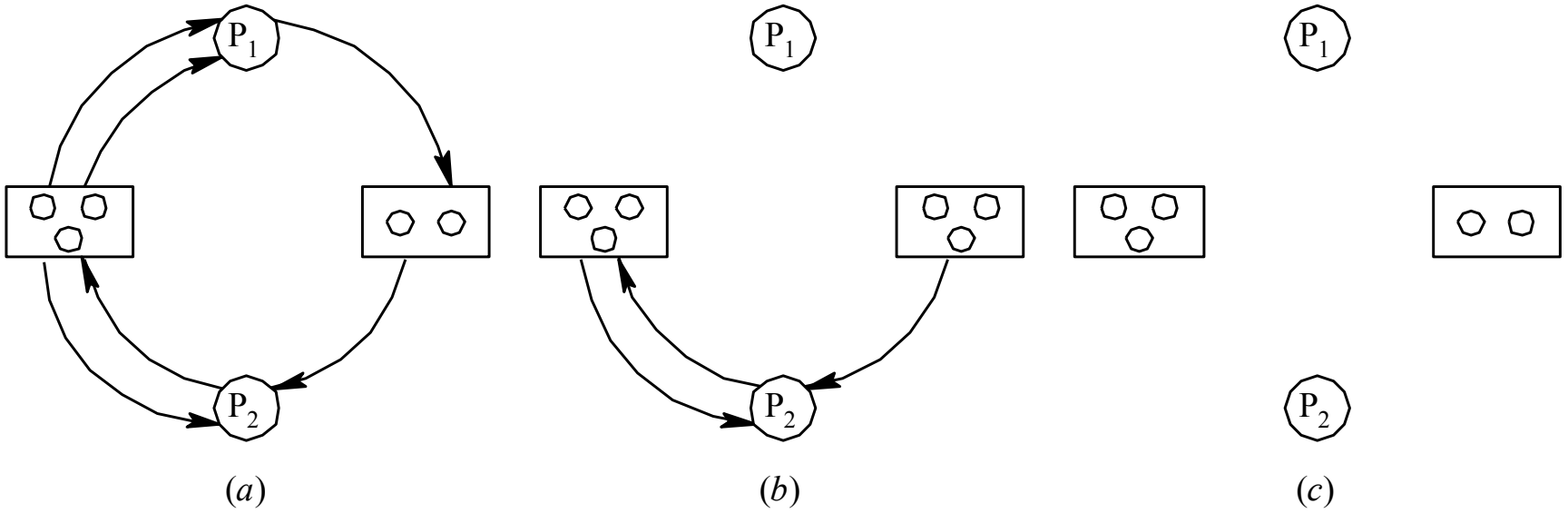


图 3-20 资源分配图的简化

3. 死锁检测中的数据结构

(1) 可利用资源向量Available，它表示*m*类资源中每一类资源的可用数目。

(2) 把不占用资源的进程(向量Allocation：=0)记入L表中， 即$L_i \cup L$。

(3) 从进程集合中找到一个$Request_i \leq Work$的进程，做如下处理：

① 将其资源分配图简化，释放出资源，增加工作向量

Work：=Work+$Allocation_i$。

② 将它记入L表中。

(4) 若不能把所有进程都记入L表中， 便表明系统状态S的资源分配图是不可完全简化的。 因此，该系统状态将发生死锁。

3.7.2 死锁的解除

(1) 剥夺资源。

(2) 撤消进程。

为把系统从死锁状态中解脱出来，所花费的代价可表示为：

$R(S)_{min}=min\{C_{ui}\}+min\{C_{uj}\}+min\{C_{uk}\}+...$

作 业

1、有五个进程A、B、C、D、E，到达时间和服务时间如下表所示，请按照非抢占式的动态高优先权优先算法，完成下表，并给出计算过程。

	进程名称	A	B	C	D	E
进程情况	到达时间	0	1	2	4	5
	服务时间	4	1	3	6	3
高优先权优先算法	等待时间					
	开始时间					
	完成时间					
	周转时间					

2、有6个进程A、B、C、D、E、F，以及三种资源a、b、c，
这6个进程对这三种资源的最大需求和已经分配的见下表，
假如系统中现有a、b、c的数目分别为1,1,1，
请按照能够避免死锁的银行家算法，给出相应的分配顺序，
并且写出每次分配后，Work (Available) 集合的变化。

进程	最大需求 a、b、c	已分配 a、b、c
A	3，2，5	1，0，0
B	4，2，4	1，0，0
C	2，2，2	0，0，1
D	1，1，2	0，0，2
E	2，2，4	1，1，0
F	1，1，3	0，0，1