计算机操作系统
Operating Systems

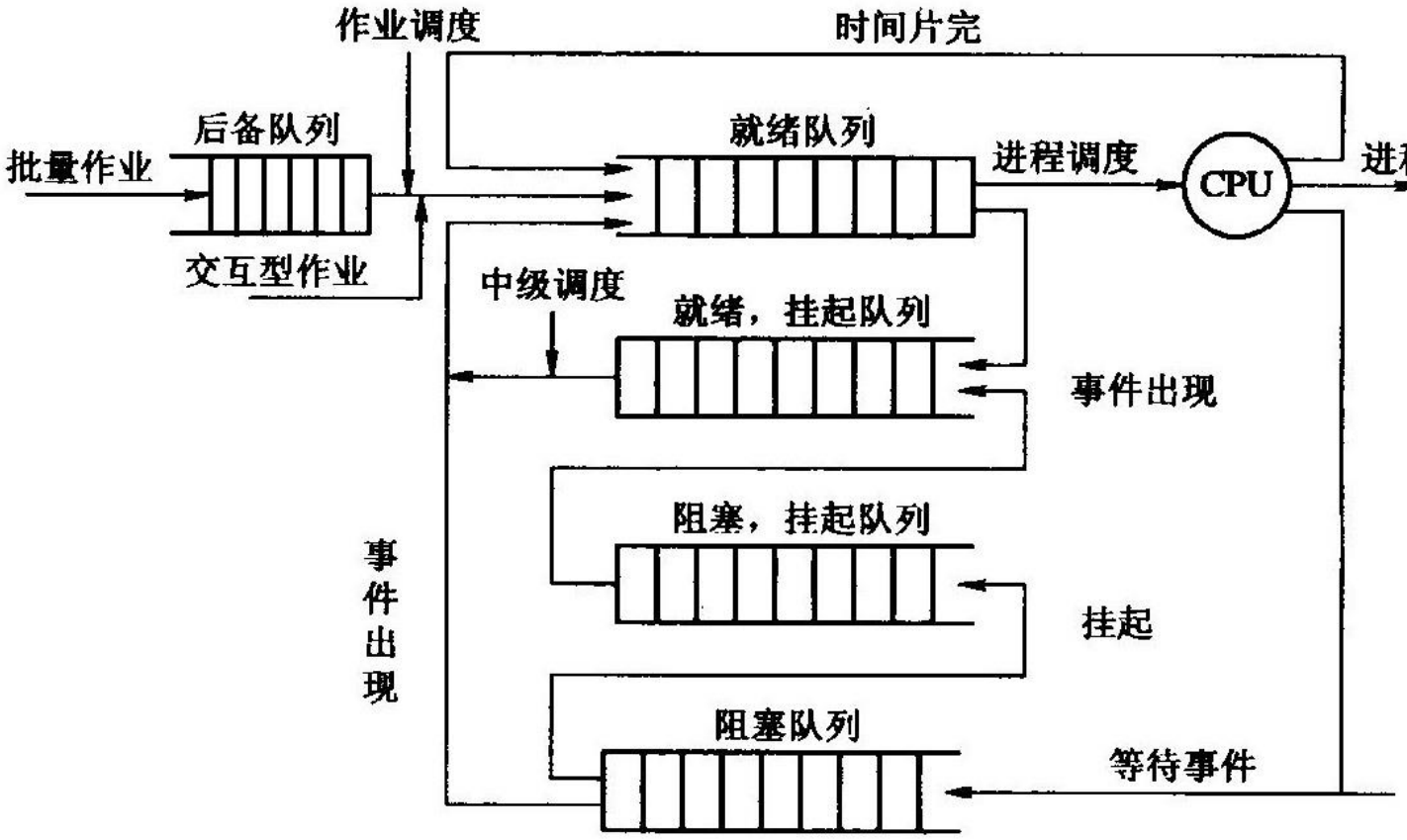
田卫东

March, 2014

第3章 处理机调度与死锁

3.1 处理机调度的基本概念

3.1.1 操作系统分配处理机的基本流程

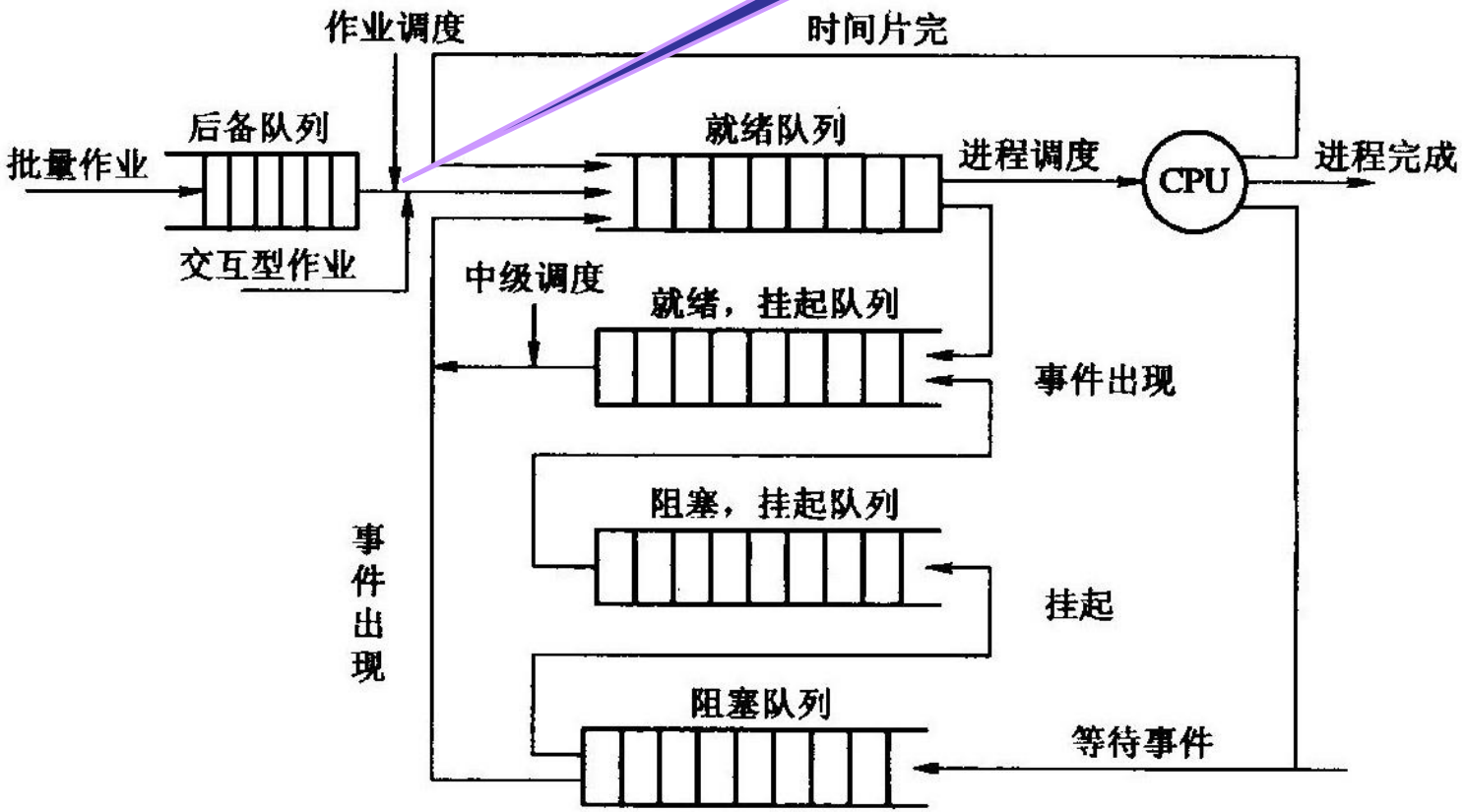


具有三级调度时的调度队列模型

3.1 处理机调度的基本概念

3.1.2 高级、中级、低级三级调度

(1) 高级调度



具有三级调度时的调度队列模型

3.1 处理机调度的基本概念

3.1.2 高、中、低三级调度

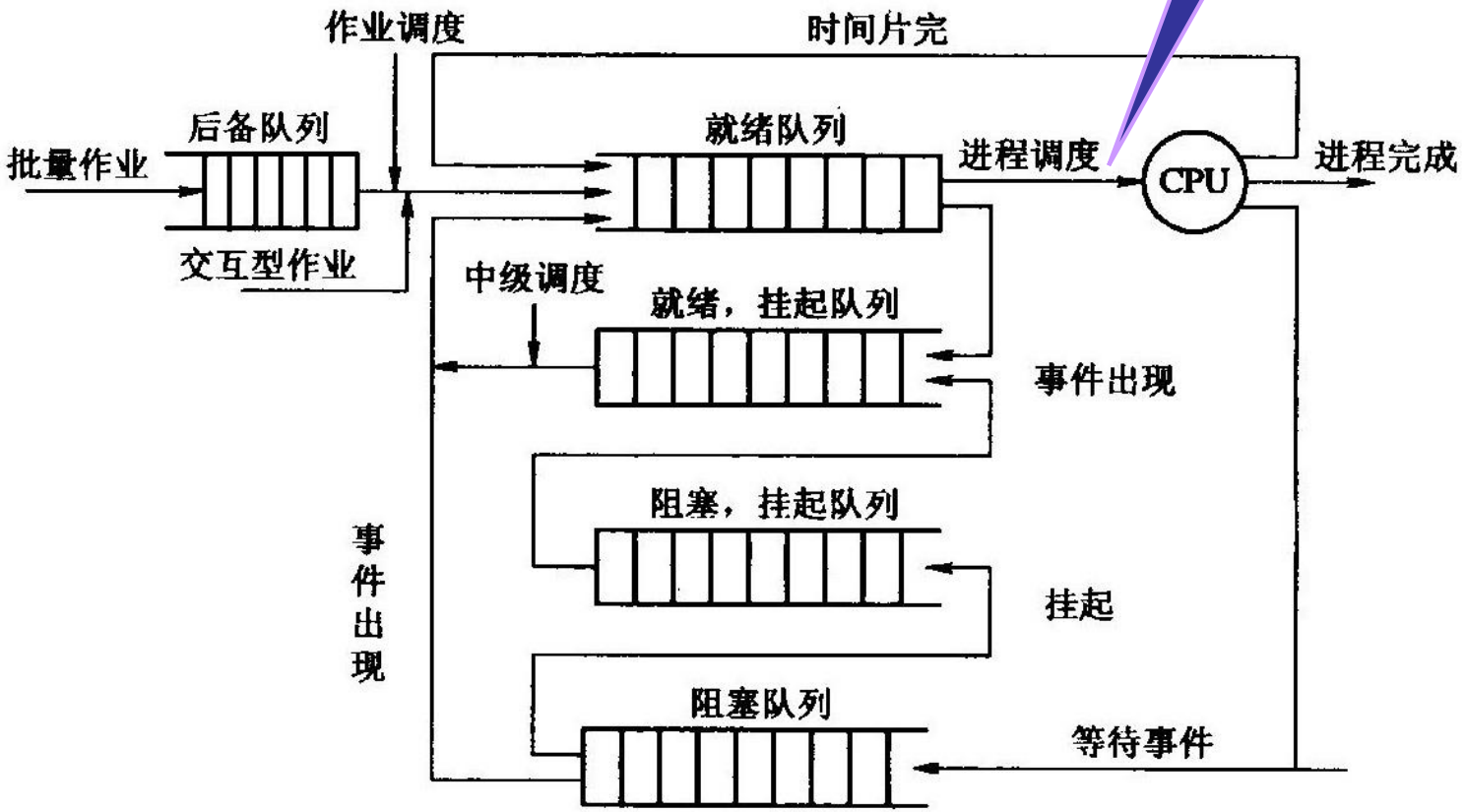
(1) 高级调度

- 高级调度的概念（Long-Term Scheduling）
  - 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、分配必要的资源，再将新创建的进程排在就绪队列上，准备执行。
  - 长程调度、作业调度、接纳调度。
  - 对象：作业
- 特点
  - 调度频率低：几分钟或几十分钟。
  - 调度算法可以很复杂

3.1 处理机调度的基本概念

3.1.2 高、中、低三级调度

(2) 低级调度



具有三级调度时的调度队列模型

3.1 处理机调度的基本概念

3.1.2 高、中、低三级调度

(2) 低级调度

- **低级调度的概念（Low-Level Scheduling）**
  - 用来决定就绪队列中的哪个进程应获得处理机，然后再由分派程序执行把处理机分配给该进程的具体操作。
  - 进程调度、短程调度。
  - 对象：就绪进程
- **特点**
  - 调度频率高：几毫秒或几十毫秒。
  - 调度算法通常简单，保证算法执行时间短

3.1 处理机调度的基本概念

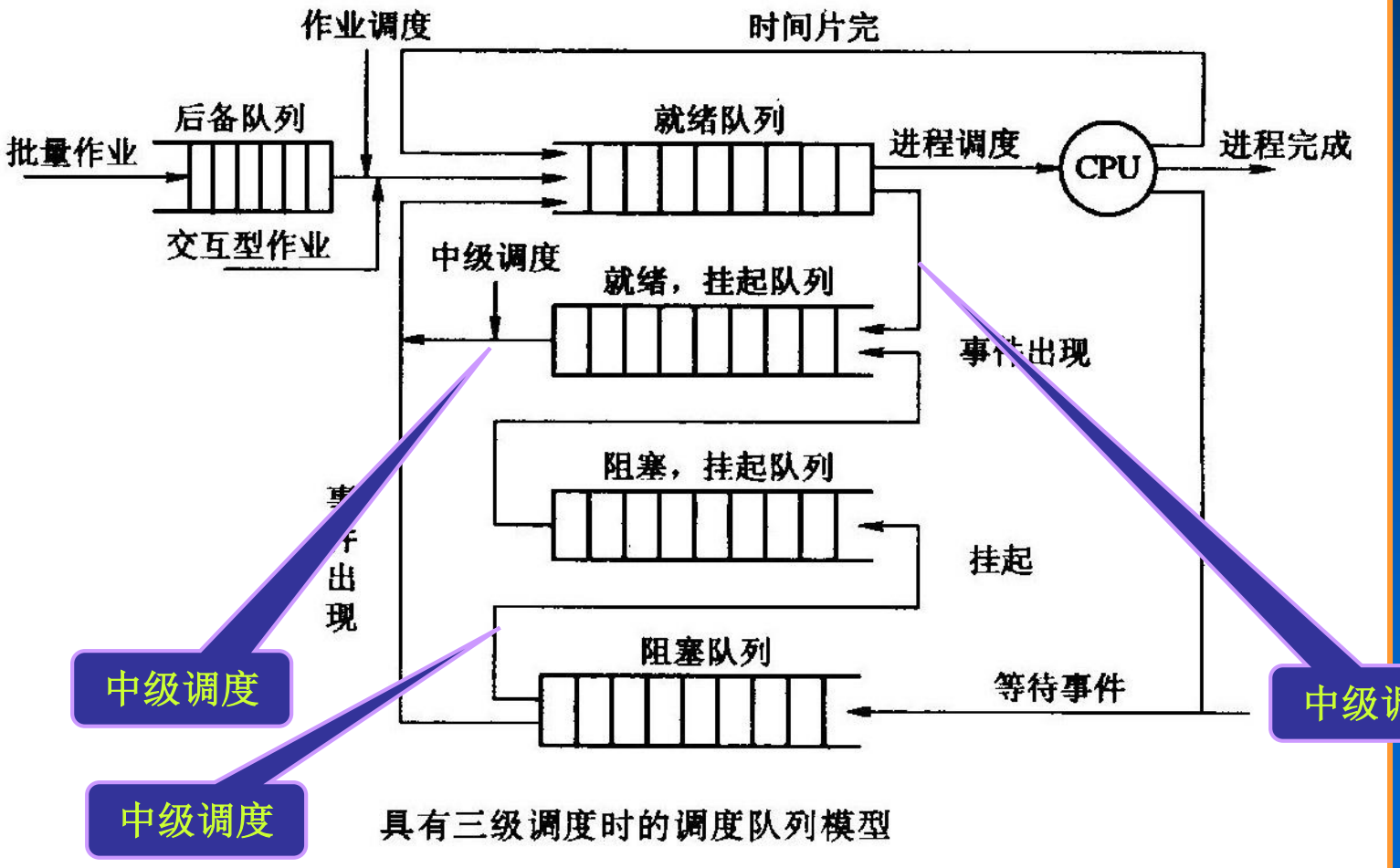
(2) 低级调度

- 调度方式：非抢占方式
  - 指当某一进程正在处理机上执行时，即使有某个更为重要或紧迫的进程进入就绪队列，仍然让正在执行的进程继续执行，直到该进程完成或发生某种事件而进入阻塞状态时，才把处理机分配给更为重要或紧迫的进程。
  - 非抢占方式又称非剥夺方式、不可剥夺方式。
  - 简单，系统开销小，实时性差。
  - 不安全：霸占CPU，造成进程”饥饿”。
- 调度方式：抢占方式
  - 指当一个进程正在处理机上执行时，若有某个更为重要或紧迫的进程需要使用处理机，则立即暂停正在执行的进程，将处理机分配给这个更重要或紧迫的进程。
  - 剥夺方式又称抢占方式、可剥夺方式。
  - 安全：不会霸占CPU。
  - 方式：优先权原则、短作业（进程）优先、时间片原则。

3.1 处理机调度的基本概念

3.1.2 高、中、低三级调度

(3) 中级调度



具有三级调度时的调度队列模型

3.1 处理机调度的基本概念

3.1.2 高、中、低三级调度

(3) 中级调度

- **中级调度的概念（Intermediate-Level Scheduling）**
  - 中级调度主要目的：为了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不再占用宝贵的内存资源，而将它们调至外存上去等待，把此时的进程状态称为就绪驻外存状态或挂起状态。
  - 当这些进程重又具备运行条件、且内存又稍有空闲时，由中级调度来决定把外存上的哪些进程，重新调入内存，并修改其状态为就绪状态，挂在就绪队列上等待进程调度。
  - 内存就绪（表示进程在内存中就绪）和外存就绪（进程在外存中就绪），内存阻塞和外存阻塞。
  - 对象：就绪进程、阻塞进程
- 特点
  - 调度频率：介于高级调度和低级调度之间。
  - 实际就是内存管理的“对换”功能

3.1 处理机调度的基本概念

3.1.3 调度队列模型

(1) 仅有进程调度的调度队列模型

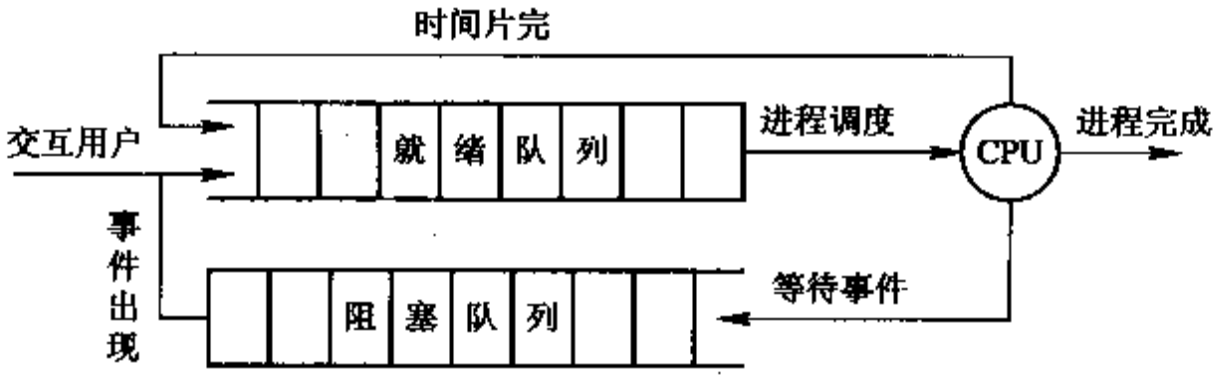


图 3－1　仅具有进程调度的调度队列模型

3.1 处理机调度的基本概念

3.1.3 调度队列模型

(2) 具有高级调度和低级调度的调度队列模型

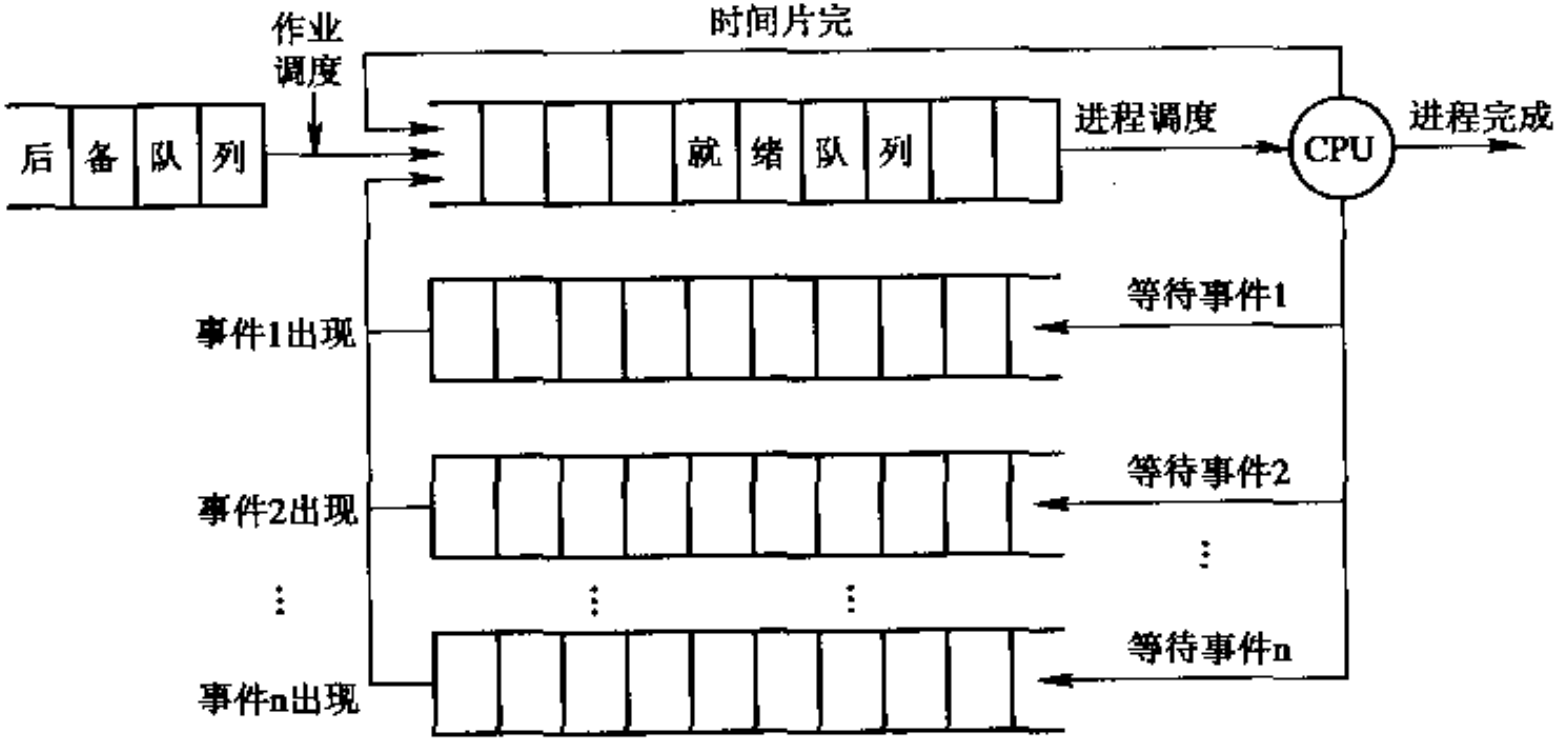
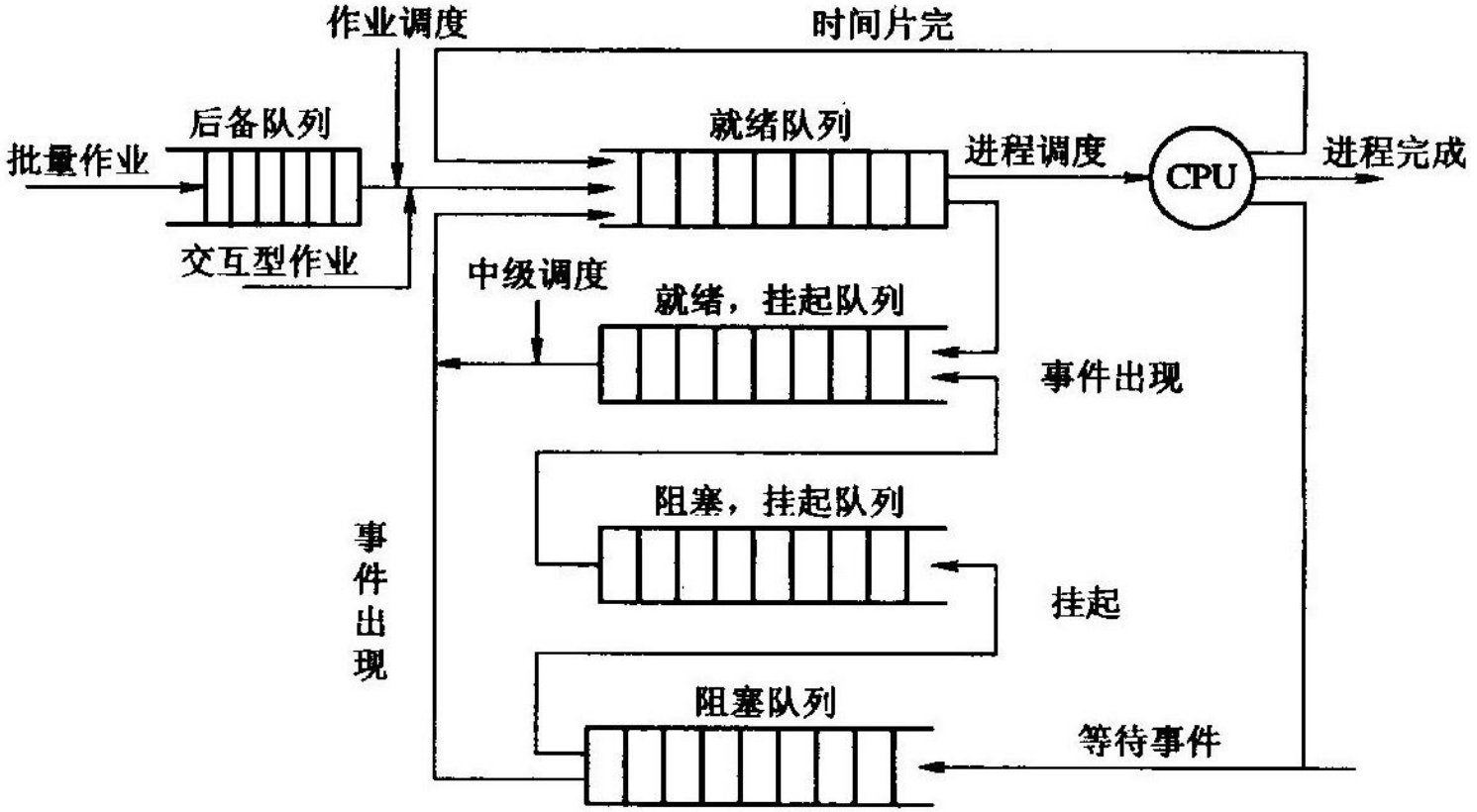


图 3－2　具有高、低两级调度的调度队列模型

3.1 处理机调度的基本概念

3.1.3 调度队列模型

(3) 同时具有高、中、低三级调度的调度队列模型



具有三级调度时的调度队列模型

3.1.3 选择调度方式和衡量调度算法的性能

(1) 面向用户的准则

- 周转时间短
  - 周转时间

    从作业提交给系统开始，到作业完成为止的时间间隔。

    包括：1)作业在后备队列的等待时间；
    2)进程在就绪队列的等待时间；
    3)进程在CPU上的执行时间；
    4)进程等待（如I/O操作）时间；
  - 平均周转时间

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

  - 平均带权周转时间。

$$W = \frac{1}{n}\sum_{i=1}^{n} \frac{T_i}{T_{si}}$$

T_i：第i个作业的周转待时间；

T_{si}：第i个作业的要求服务时间；

3.1 处理机调度的基本概念

3.1.3 选择调度方式和衡量调度算法的性能

(1) 面向用户的准则

- 响应时间快

响应时间：用户从提交键盘命令开始，到系统首次给出响应为止的时间。

- 截止时间保证

截止时间：作业/进程开始或结束的最晚时间。

(2) 面向系统的准则

- 系统吞吐量高

吞吐量：系统单位时间内完成的作业/进程数量。

- 处理机利用率好
- 资源利用均衡

3.2 调度算法

- 实质是一种资源分配方法，因而调度算法是指：根据系统的资源分配策略所规定的资源分配算法。
- 对于不同的系统和系统目标，通常采用不同的调度算法，例如，在批处理系统中，为了照顾为数众多的短作业，应采用短作业优先的调度算法．又如在分时系统中，为了保证系统具有合理的响应时间，应采用轮转法进行调度。
- 目前存在的多种调度算法中，有的算法适用于作业调度，有的算法适用于进程调度；但也有些调度算法既可用于作业调度，也可用于进程调度。

3.2 调度算法

3.2.1 先来先服务调度算法(FCFS)

(1) 基本设计思想

- FCFS概述
  - 既可用于作业调度，也可用于进程调度。
  - 用于作业调度：每次调度都是从后备作业队列中，选择一个或多个最先进入该队列的作业，将它们调入内存，为它们分配资源、创建进程，然后放入就绪队列。
  - 用于进程调度：则每次调度是从就绪队列中，选择一个最先进入队列的进程，为之分配处理机，使之投入运行。

3.2 调度算法

3.2.1 先来先服务调度算法(FCFS)

(2) 特点

- 本质

  仅考虑“到达时间”；

- 特点
  - 实现简单；
  - 貌似公平；
  - 实际上，对短作业不公平。

3.2 调度算法

3.2.2 短作业/短进程优先调度算法(SJF/SPF)

(1) 基本设计思想

- SJF/SPF概述
  - 既可用于作业调度(SJF)，也可用于进程调度(SPF)。
  - 用于作业调度：每次调度都是从后备作业队列中，选择一个要求服务时间（执行时间）最短的作业，将它们调入内存，为它们分配资源、创建进程，然后放入就绪队列。
  - 用于进程调度：则每次调度是从就绪队列中，选择一个执行时间最短的进程，为之分配处理机，使之投入运行。

3.2 调度算法

3.2.2 短作业/短进程优先调度算法(SJF/SPF)

(2) 特点

- 本质

  仅考虑“执行时间”；

- 特点
  - 实现困难：估算执行时间很难；
  - 有利于短作业
  - 对长作业不公平。

3.2 调度算法

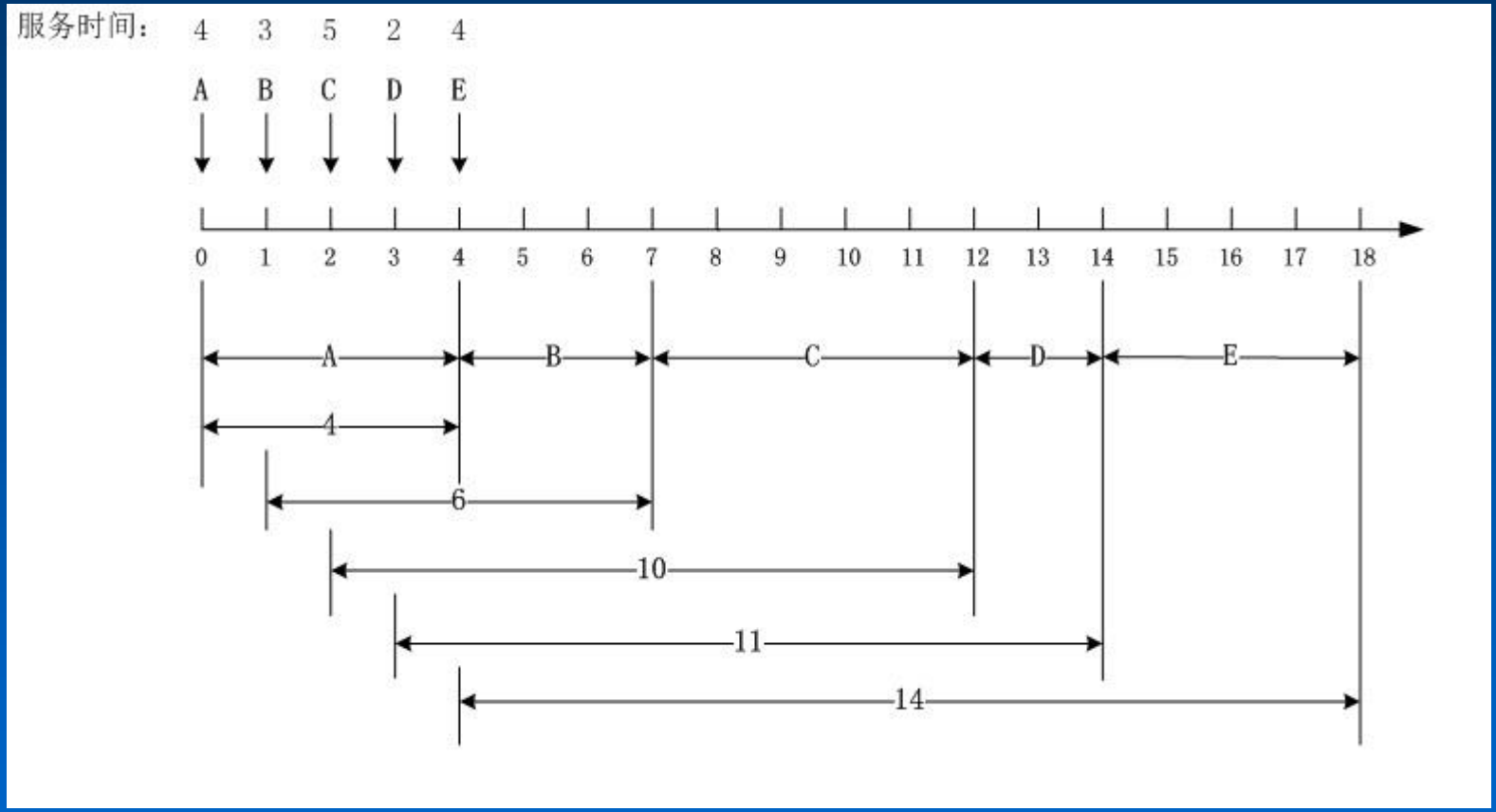
3.2.2 短作业/短进程优先调度算法(SJF/SPF)

(3) 算法对比实例1

- 5进程A,B,C,D,E分别于0,1,2,3,4时刻到达系统，要求服务时间分别为4,3,5,2,4, 请计算：
(1)采用FCFS时进程执行顺序和平均周转时间；
(2)采用SPF时进程执行顺序和平均周转时间；

3.2 调度算法

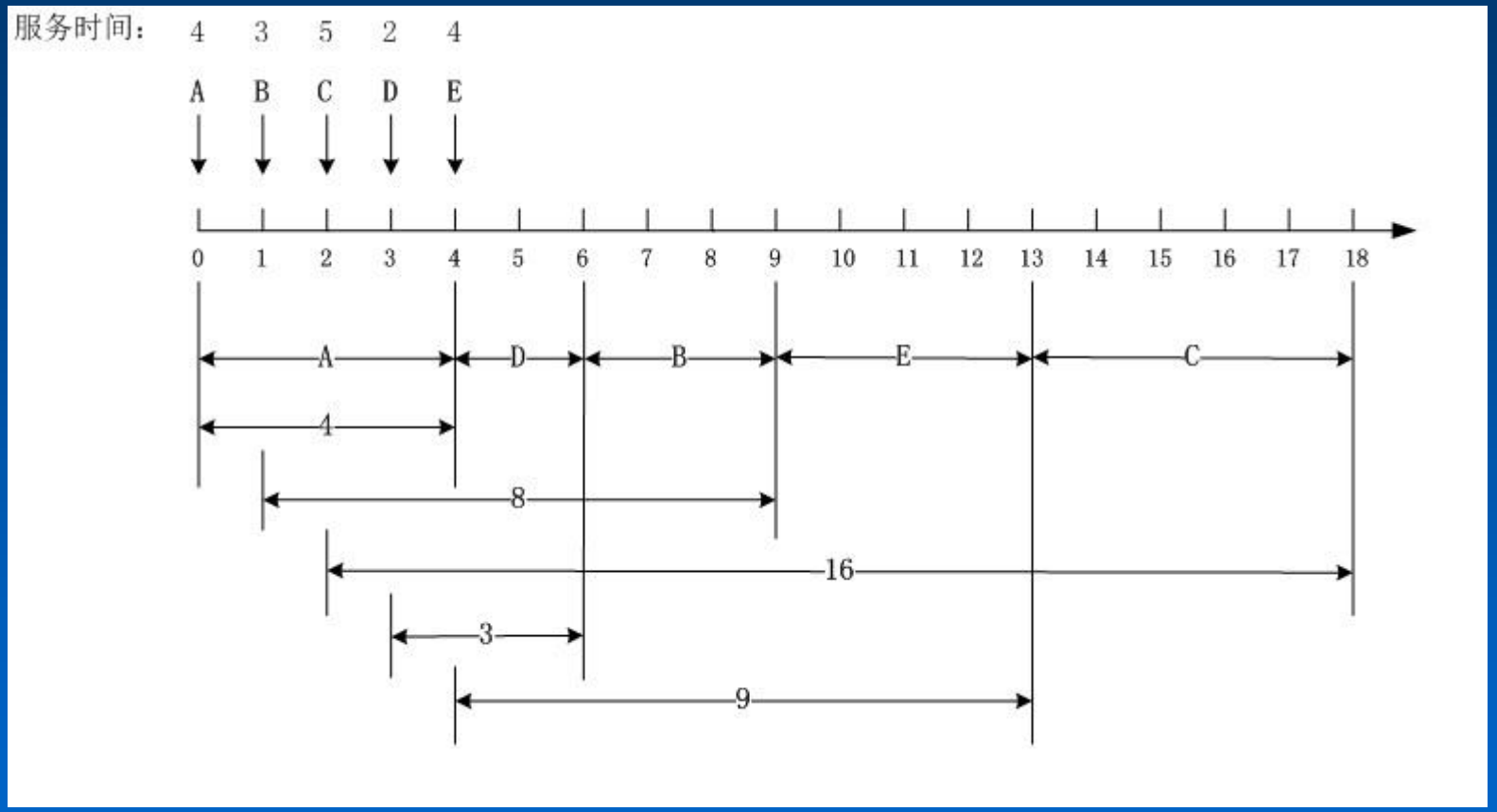
(1) FCFS



- 进程执行顺序：A→B→C→D→E
- 平均周转时间：T = (4+6+10+11+14) / 5 = 9
- 平均带权周转时间：W=(4/4+6/3+10/5+11/2+14/4) / 5 = 2.8

3.2 调度算法

(2) SPF



- 进程执行顺序：A→D→B→E→C
- 平均周转时间：T= (4+8+16+3+9) / 5 = 8
- 平均带权周转时间：W=(4/4+8/3+16/5+3/2+9/4) / 5 ≈ 2.12

3.2 调度算法

3.2.3高优先权优先调度算法(FPF)

(1) 基本设计思想

- FPF概述
  - 既可用于作业调度，也可用于进程调度。
  - 用于作业调度时：系统将从后备队列中选择若干个优先权最高的作业，装入内存。
  - 用于进程调度时：该算法是把处理机分配给就绪队列中优先权最高的进程。分为非抢占式优先权算法和抢占式优先权调度算法。

3.2 调度算法

3.2.3高优先权优先调度算法(FPF)

(2) 优先权

- 优先权：算法的核心
  - 反映作业/进程执行时的迫切程度，是对调度所考虑的实际因素的算法抽象。
  - 通常用1个整型数来表示。
- 进程的优先权
  - 抢占式优先权（进程调度）：高优先权进程到达时，立刻停止低优先权进程的执行，让高优先权进程执行。
  - 非抢占式优先权（进程调度）：高优先权作业进程到达时，须等待低优先权进程执行完毕或主动释放CPU。

3.2 调度算法

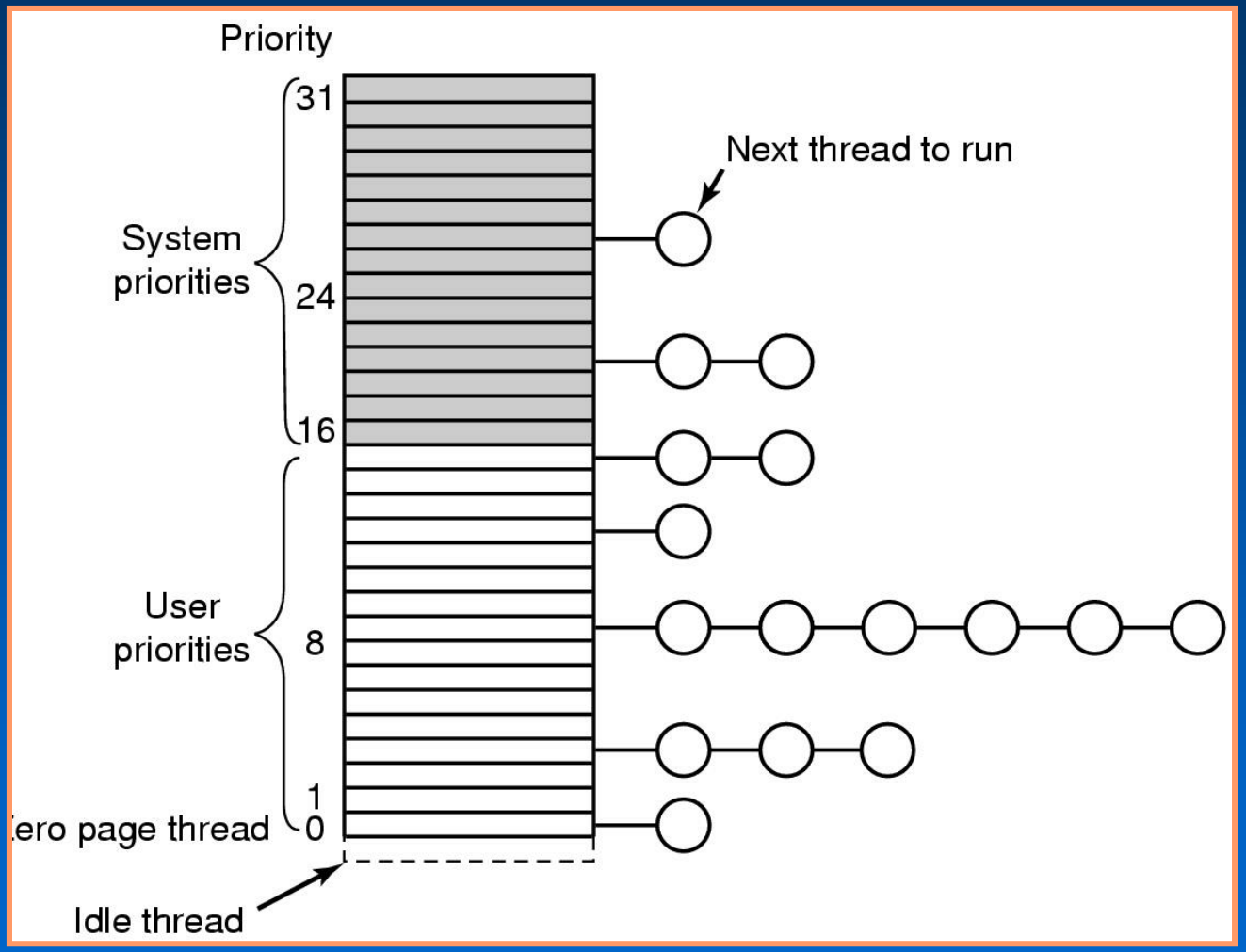
3.2.3高优先权优先调度算法(FPF)

(2) 优先权

- 优先权的确定
  - 静态优先权：进程创建时确定（根据进程类型、资源需求和用户要求等），直到进程执行结束，保持不变。
  - 动态优先权：进程创建时确定（根据进程类型、资源需求和用户要求等）初始优先权，在进程执行过程中，可以发生变化。

(3) 优先权示例：Windows 2000/XP的32级线程优先权

Windows 2000支持32级线程优先级



3.2 调度算法

3.2.4 高响应比优先调度算法

(1) 基本设计思想

- 概述
  - 用于作业调度。
  - 系统将从后备队列中选择若干个响应比(优先权)最高的作业，装入内存，投入运行。

$$\text{优先权} = \frac{\text{响应时间}}{\text{要求服务时间}}$$

$$\text{优先权} = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}} = 1 + \frac{\text{等待时间}}{\text{要求服务时间}}$$

响应比*Rp*

  - 核心：对等待作业以一定速率α提高其优先权。

3.2 调度算法

3.2.4 高响应比优先调度算法

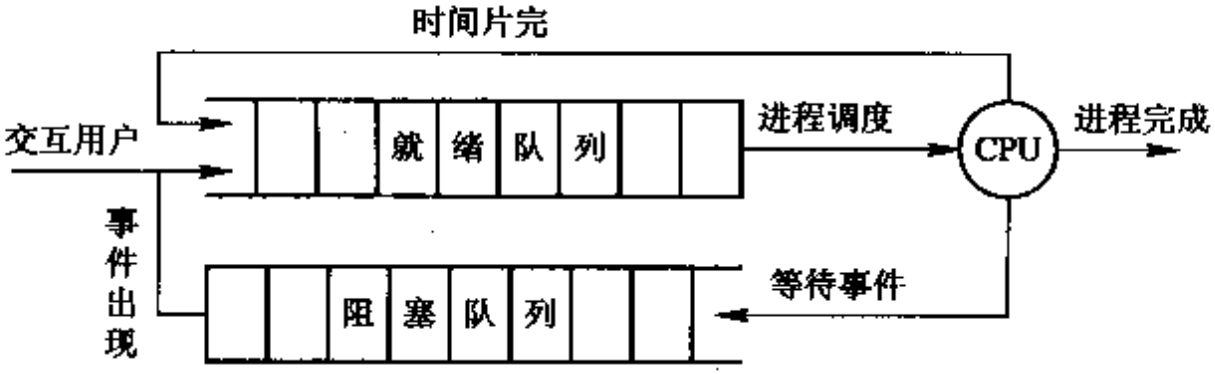
(2) 特点

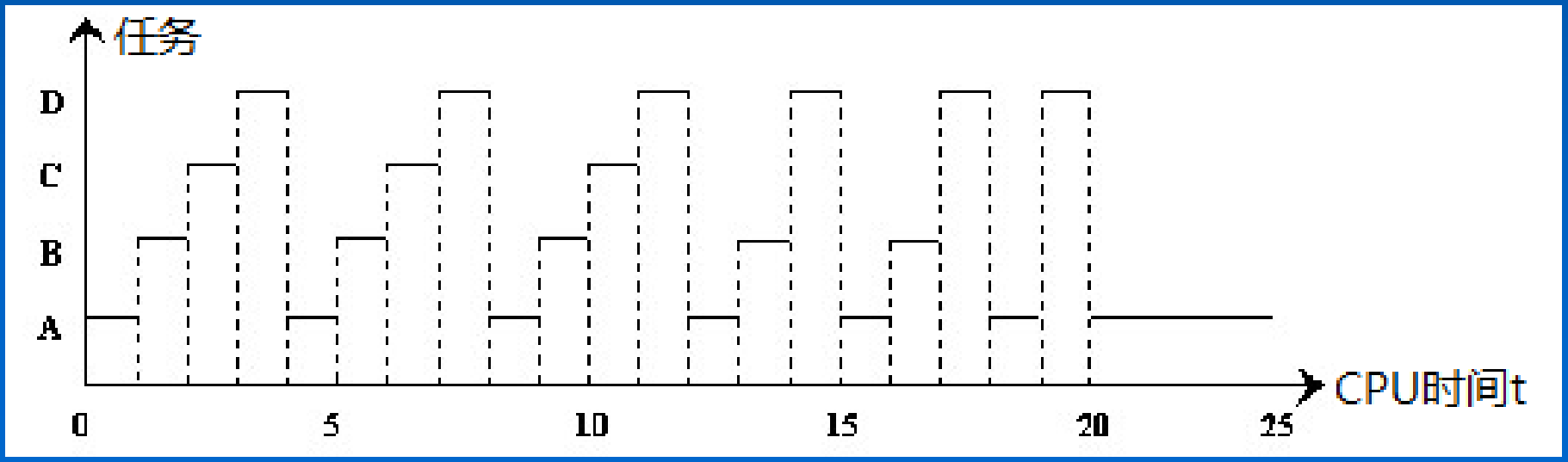
- 如果作业的等待时间相同，则要求服务的时间愈短，其优先权愈高，因而该算法有利于短作业。
- 当要求服务的时间相同时，作业的优先权决定于其等待时间，等待时间愈长，其优先权愈高，因而它实现的是先来先服务。
- 对于长作业，作业的优先级可以随等待时间的增加而提高，当其等待时间足够长时，其优先级便可升到很高，从而获得处理机。
- 既照顾了短作业，又考虑了作业到达的先后次序，不会使长作业长期得不到服务。因此，该算法实现了一种较好的折衷。
- 须做响应比计算，会增加系统开销

3.2 调度算法

3.2.5 时间片轮转调度算法(RR)

(1) 基本设计思想





3.2 调度算法

3.2.5 时间片轮转调度算法(RR)

(1) 基本设计思想

- 概述
  - 用于作业调度或进程调度。
  - 在早期的时间片轮转法中，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。
  - 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。
  - 保证响应时间：就绪队列中的所有进程，在给定时间内，均能获得一个时间片的处理机执行时间。换言之，系统能在给定的时间内，响应所有用户的请求。
  - 时间片的大小从几毫秒 到几百毫秒。

3.2 调度算法

3.2.5 时间片轮转调度算法(RR)

(2) 时间片（Time Slice）

- 时间片选择
  - 固定时间片。
  - 可变时间片。
- 时间片大小选择
  - 不可太大：影响最大响应时间：

    $T=nq$；

    其中，n为进程数量，q为时间片大小。
  - 不可太小：

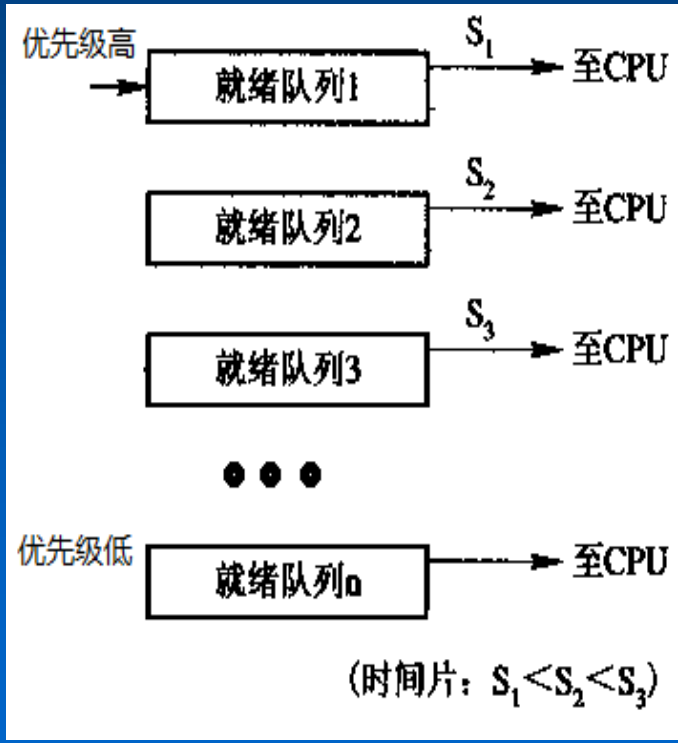
    调度开销，增加周转时间；

3.2 调度算法

3.2.6 多级反馈队列调度算法(MFQ)

(1) 基本设计思想

- 算法概述
- 设置多个就绪队列，为各队列赋予不同的优先级。第1个队列的优先级最高，其余队列优先权逐个降低。
- 赋予各个队列中进程执行时间片的大小各不相同：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。
- 仅当第1队列空闲时，调度程序才调度第2队列中的进程运行；仅当第1～(i-1)队列均空时，才会调度第i队列中的进程运行。
- 如果处理机正在第i队列中为某进程服务时，又有新进程进入优先权较高的队列(第1～(i-1)中的任何一个队列)，则此时新进程将抢占正在运行进程的处理机（正在运行进程被放回原所在队列末尾）。



3.2 调度算法

3.2.6 多级反馈队列调度算法

(1) 基本设计思想

- 算法概述
- 对新创建进程，首先将它放入第1队列末尾，按FCFS原则排队等待调度。
- 当轮到该进程执行时，如它能在该时间片内完成，则结束；如果未完成，调度程序便将该进程转入第2队列的末尾，再同样按FCFS原则等待调度执行；
- 如果它在第2队列中运行1个时间片后仍未完成，再依次将它放入第3队列…。如此下去，当1个长作业（进程）从第1队列依次降到第n队列后，在第n队列中便采取按时间片轮转的方式运行。

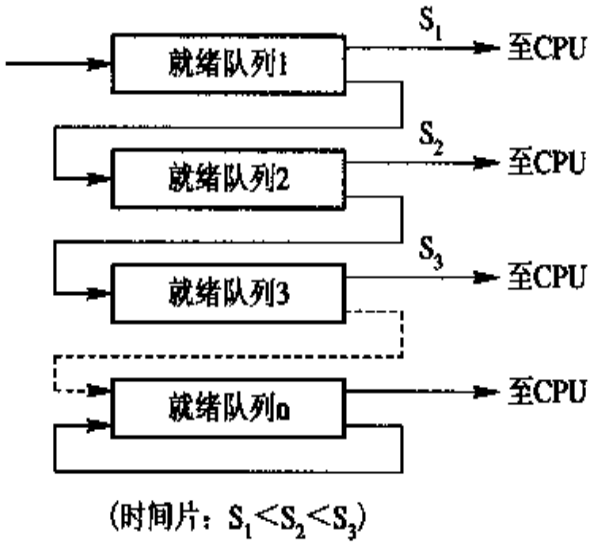


图 3－5　多级反馈队列调度算法

3.2 调度算法

3.2.6 多级反馈队列调度算法

(2) 特点

- 避免了事先计算各种进程所需的执行时间。
- 具有较好的性能，能较好地满足各种类型用户需要：
  - 终端型作业用户：终端型作业大多属于交互型作业，作业通常较小，系统只要能使这些作业（进程）在第一队列所规定的时间片内完成，便可使终端型作业用户都感到满意。
  - 短批处理作业用户：对于很短的批处理型作业，开始时像终端型作业一样，如果仅在第一队列中执行一个时间片即可完成，便可获得与终端型作业一样的响应时间。对于稍长的作业，通常也只需在第二队列和第三队列各执行一个时间片即可完成，其周转时间仍然较短。
  - 长批处理作业用户： 对于长作业，它将依次在第1, 2, …,n个队列中运行，然后再按轮转方式运行，用户不必担心其作业长期得不到处理。

3.3 实时调度

- 实时系统中的调度问题

  实时系统中都存在着若干个实时进程或任务，它们用来反应或控制某个（些）外部事件，往往带有某种程度的紧迫性，因而对实时系统中的调度提出了某些特殊要求，前面所介绍的多种调度算法，并不能很好地满足实时系统对调度的要求。为此，引入一种新的调度，即实时调度。

3.3 实时调度

3.3.1 实现实时调度的基本条件

- 可调度条件

  对于m个实时任务，处理时间为C_i,周期时间为P_i，则系统是可调度的，如果满足下列条件：

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

- 示例1：可调度：
  A: 周期：1s, 处理时间：70ms
  B: 周期：1s, 处理时间：90ms
- 示例2：不可调度
  A: 周期：1s, 处理时间：700ms
  B: 周期：1s, 处理时间：900ms

3.3 实时调度

3.3.2 实时调度算法的分类

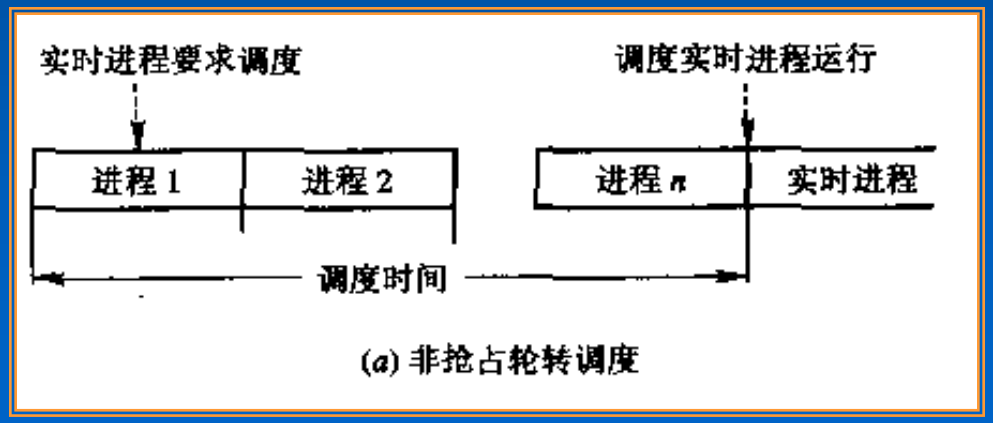
- 根据实时任务性质
  - 硬实时调度算法：严格实时
  - 软实时调度算法：非严格实时
- 根据调度方式
  - 非抢占调度算法
  - 抢占调度算法
- 根据调度时间
  - 静态调度：在进程执行前，调度程序便已经决定了各进程间的执行顺序。
  - 动态调度算法：在进程的执行过程中，由调度程序届时根据情况临时决定将哪一进程投入运行；

3.3 实时调度

3.3.2 实时调度算法的分类

(1) 非抢占式调度算法

- 非抢占式轮转调度算法
  - 将所有实时任务排列成队列，轮流投入运行；当任务完成，就重新排到队列末尾；
  - 调度时选择队首进程；
  - 实时响应时间：数秒-数十秒



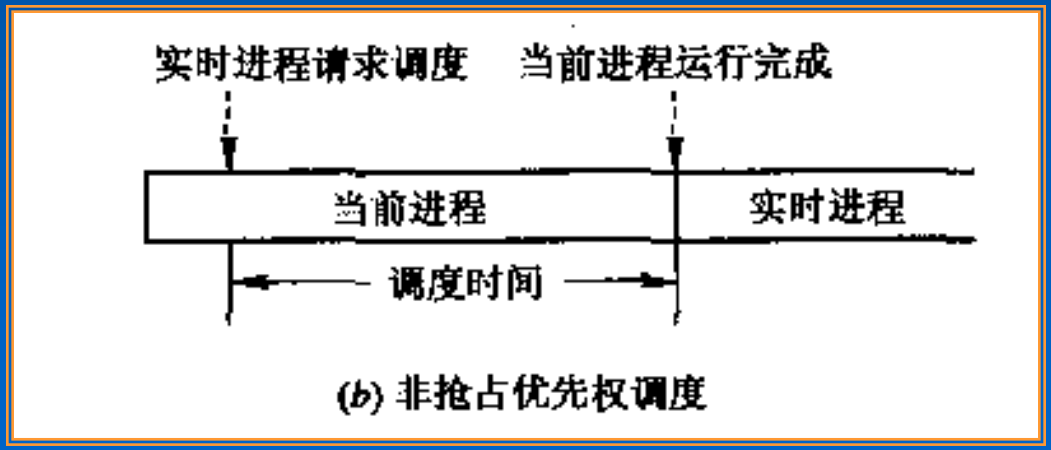
(a) 非抢占轮转调度

3.3 实时调度

3.3.2 实时调度算法的分类

(1) 非抢占式调度算法

- 非抢占式优先调度算法
  - 为所有实时任务设定优先权；高优先权进程到达队列，排到队首，以待当前任务执行完毕；
  - 调度时选择队首进程；
  - 实时响应时间：数百毫秒-数秒



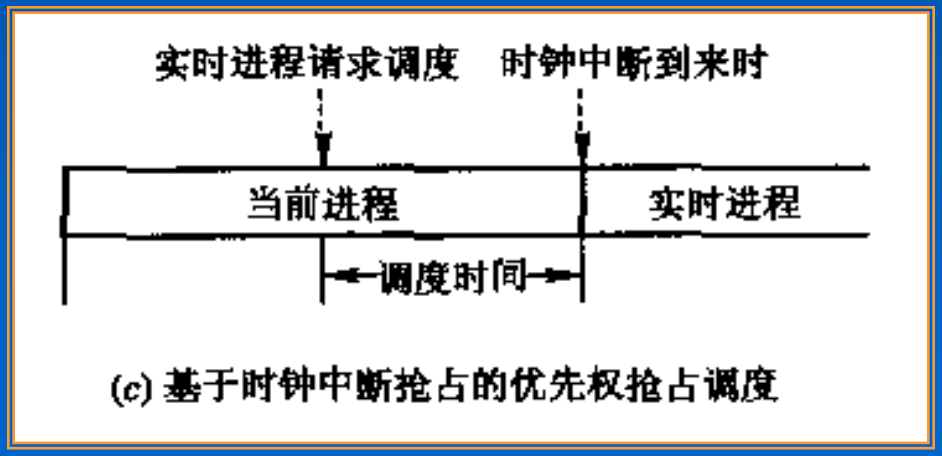
(b) 非抢占优先权调度

3.3 实时调度

3.3.2 实时调度算法的分类

(2) 抢占式调度算法

- 基于时钟中断的抢占式优先权调度算法
  - 为所有实时任务设定优先权；
  - 高优先权进程到达队列，排到队首，以待时钟中断到达时，调度到CPU上执行；
  - 调度时选择队首进程；
  - 实时响应时间：数毫秒-数十毫秒



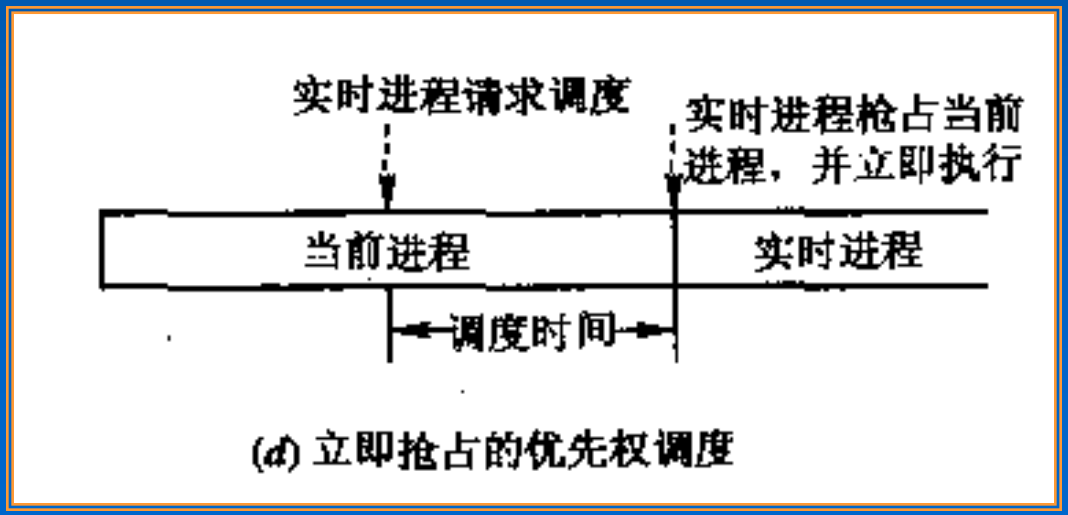
(c) 基于时钟中断抢占的优先权抢占调度

3.3 实时调度

3.3.2 实时调度算法的分类

(2) 抢占式调度算法

- 立即抢占的优先权调度算法
  - 为所有实时任务设定优先权；
  - 高优先权进程到达队列，只要当前进程不在临界区，则立刻抢占CPU；
  - 调度时选择队首进程；
  - 实时响应时间：数百微秒-数毫秒



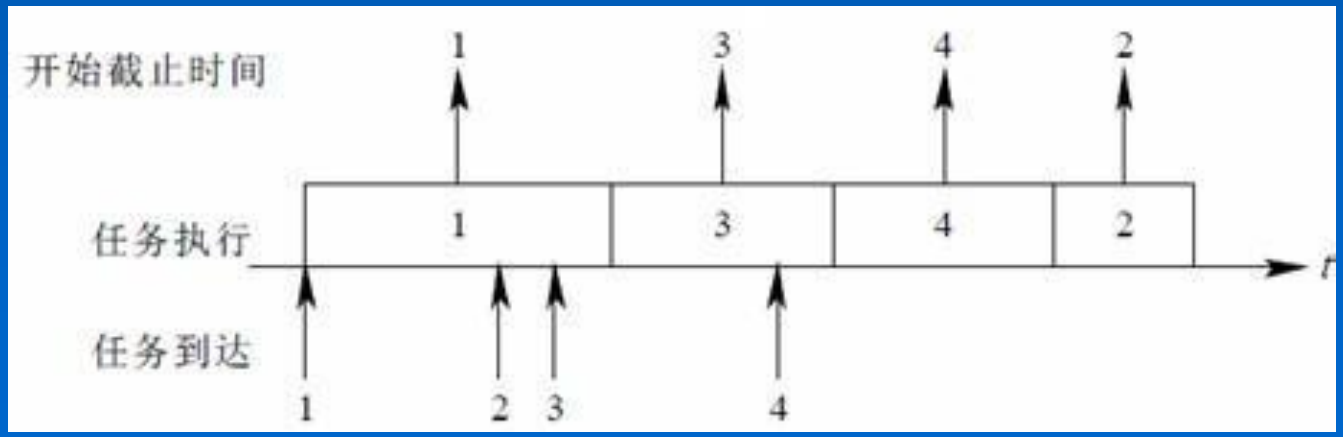
(d) 立即抢占的优先权调度

3.3 实时调度

3.3.3 两个典型的实时调度算法

(1) 最早截止时间优先调度算法(EDF)

- 根据任务的开始截止时间来确定任务的优先级。
- 截止时间愈早，其优先级愈高。
- 该算法要求在系统中保持一个实时任务就绪队列，该队列按各任务截止时间的早晚排序，具有最早截止时间的任务排在队列的最前面。
- 调度程序在选择任务时，总是选择就绪队列中的第一个任务，为之分配处理机，使之投入运行。



3.3 实时调度

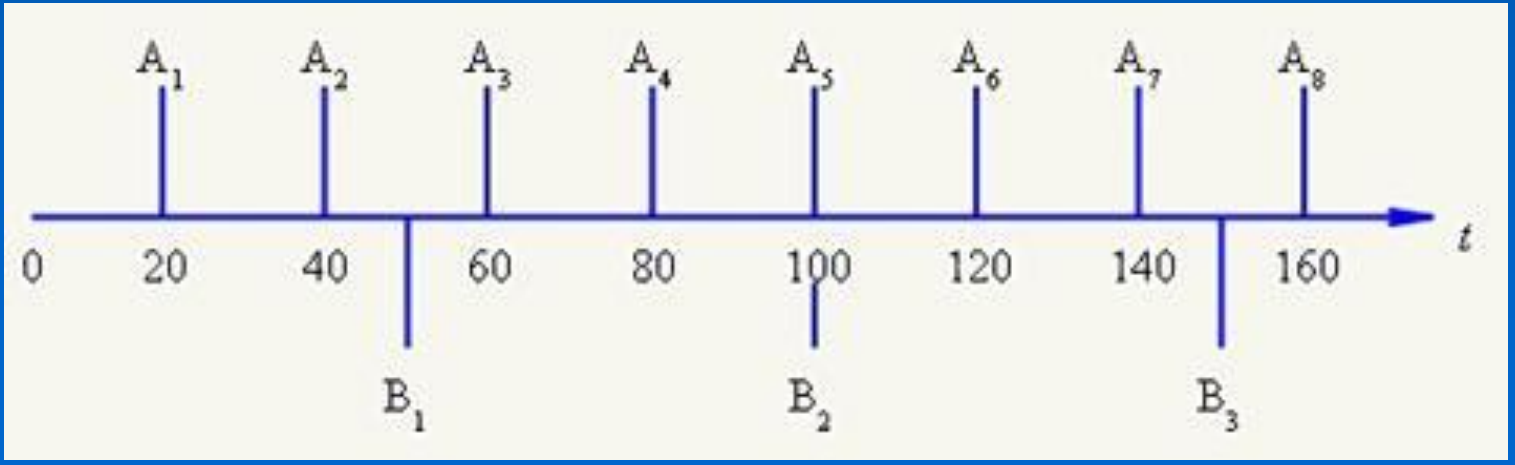
3.3.3 两个典型的实时调度算法

(2) 最低松弛度优先调度算法(LLF)

- 根据任务的紧急程度（松弛度）来确定任务的优先级。

  松弛度 = 必须完成时间 - 本身运行时间 - 当前时间
- 调度程序在选择任务时，总是选择就绪队列中紧急程度（松弛度）最大任务，为之分配处理机，使之投入运行。
- 算法示例：

  周期性任务A和B，执行时间分别为10ms和25ms，周期分别为20ms和50ms。



● 调度过程：

(1) 0ms时刻：

$A_1 = 20 - 10 - 0 = 10$

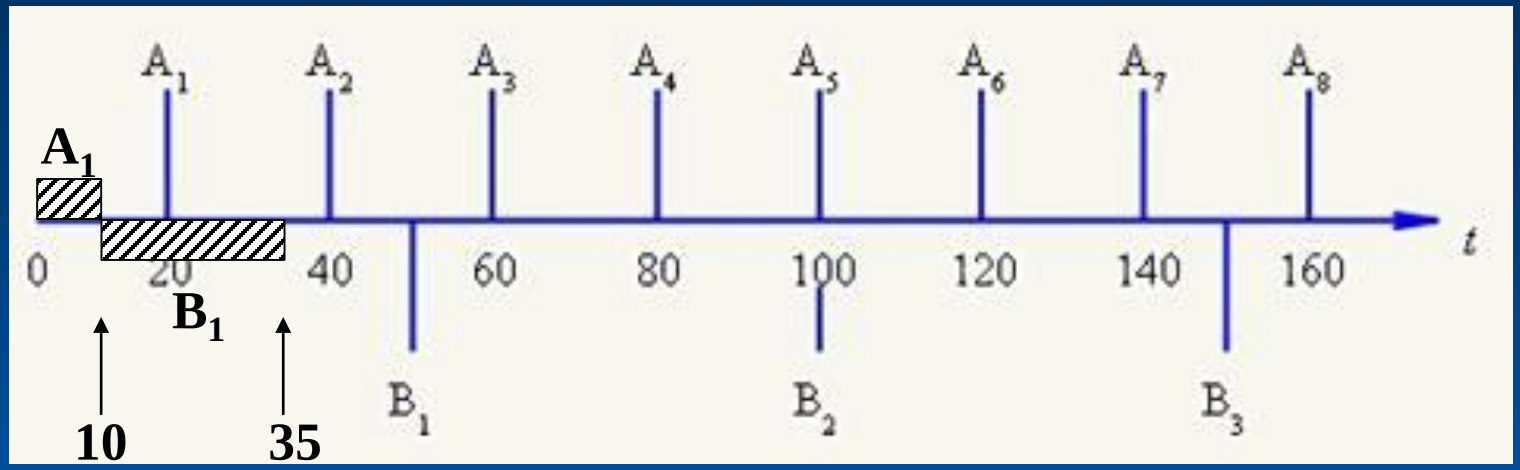
$B_1 = 50 - 25 - 0 = 25$

● 调度过程：

(2) 10ms时刻：

$A_2 = 40 - 10 - 10 = 20$

$B_1 = 50 - 25 - 10 = 15$



● 调度过程：

(3) 30ms时刻：

$A_2 = 40 - 10 - 30 = 0$

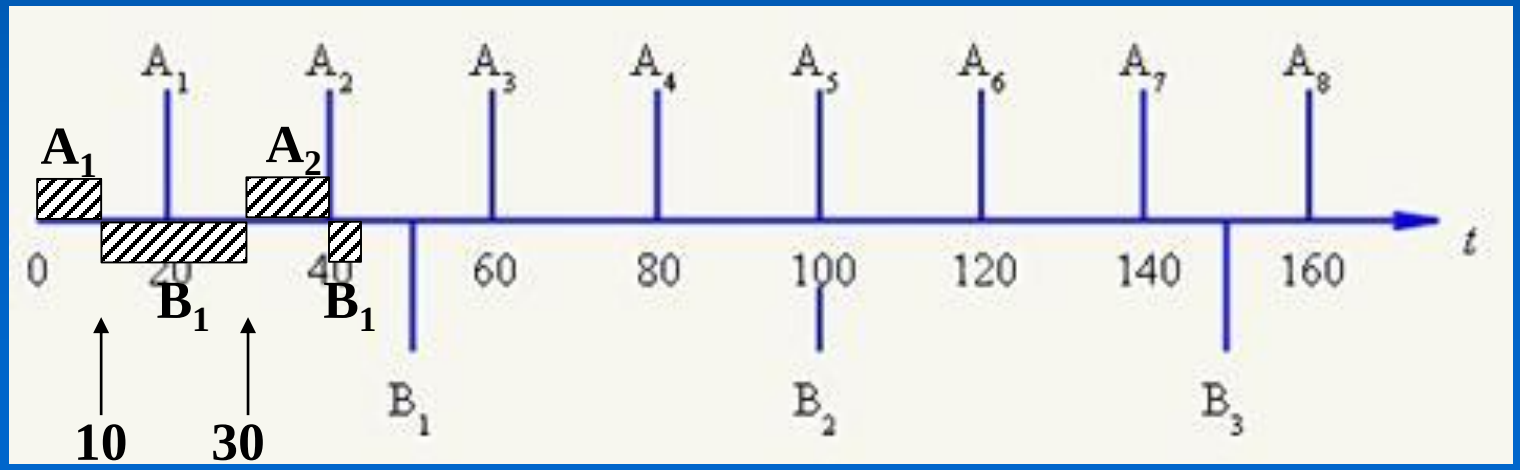
$B_1 = 50 - 5 - 30 = 15$

● 调度过程：

(4) 40ms时刻：

$A_3 = 60 - 10 - 40 = 10$

$B_1 = 50 - 5 - 40 = 5$



3.3 实时调度

3.3.4 优先级倒置问题

(1) 优先级倒置

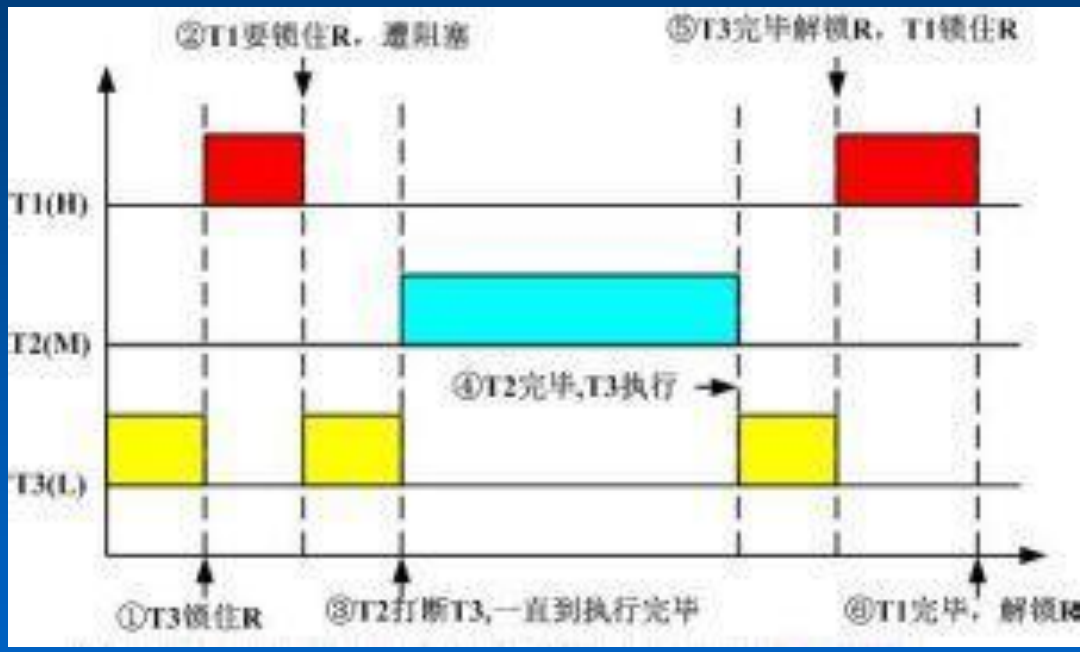
- 正常：
  高优先级进程先执行，低优先级进程后执行
- 特殊情况：
  低优先级进程先执行，高优先级进程后执行；
- 后果：
  高优先级进程执行被延迟。

3.3 实时调度

3.3.4 优先级倒置问题

(2) 优先级倒置实例

```
T1(H):
      wait(mutex);
      critical sction;
      signal(mutex);
T2(M):
      remainder section ;
T3(L):
      wait(mutex);
      critical section ;
      signal(mutex);
```



- 结果：
  低优先级T2比高优先级T1先执行，使得T1执行延迟。

3.3 实时调度

3.3.4 优先级倒置问题

(3) 优先级倒置的解决方法

- 优先权继承：

  当低优先级进程阻塞高优先级进程，将低优先级进程的优先级提高到被它被阻塞进程的最高优先级进程的优先级。