第五章 处理机调度

5.1 分级调度与调度目标

5.1.1 作业的概念

5.1.2 处理机调度的层次

5.1.3 进程调度方式与时机

5.1.4 调度算法选择依据与性能评价

5.1.1 作业的概念

1．作业和作业步

▶作业：我们把计算机业务处理过程中，从程序输入开始到输出建档，用户要求计算机所做的有关该次业务的全部工作称为一个作业。

- ♦作业由若干相互独立又相互联系的加工步骤顺序组成。每一个加工步骤称为一个作业步。
- ♦作业用于早期批处理系统和现在的大型机、巨型机系统。

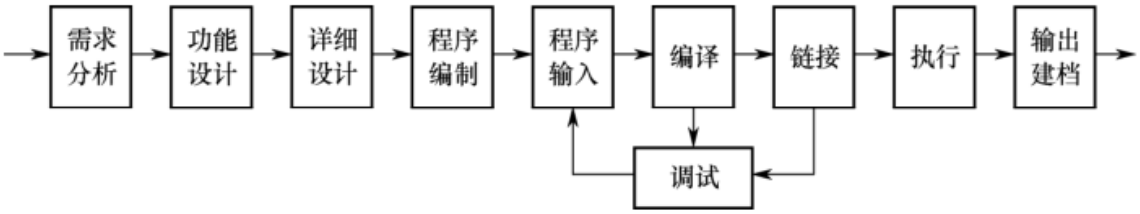


图 5-1　计算机业务的处理流程

5.1.1 作业的概念

2. 作业说明书和作业控制块

- 在批处理系统中，一个作业提交给系统之后，系统还需要获知每个作业步的先后顺序及处理要求等信息。
- 作业在提交系统时，需要编制作业说明书，一起提交给系统。
- 作业说明书体现了用户的控制意图，由作业说明书在系统中生成一个称为作业控制块的数据结构，从而达到管理和调度作业的目的。
- 作业控制块保存了系统对作业进行管理和调度所需的全部信息，主要包含作业基本描述、作业控制描述和资源要求描述。

5.1.1 作业的概念

- 作业由程序、数据和作业说明书三部分组成。
- 从系统的角度来看，作业是一个比进程更广的概念。一个作业总是由一个以上的进程组成。
  - 首先，系统为每个作业创建一个根进程；
  - 然后，按照作业控制语句的要求，系统或根进程为每个作业步创建至少一个相应的子进程；
  - 最后，为各个子进程分配资源及调度各子进程执行以完成作业要求的任务。

5.1.2 处理机调度的层次

- 处理机调度按照层次可以分为三级：高级调度、中级调度和低级调度。
- 用户作业从进入系统成为后备作业开始，直到运行结束退出系统为止，均需经历不同层次的调度。

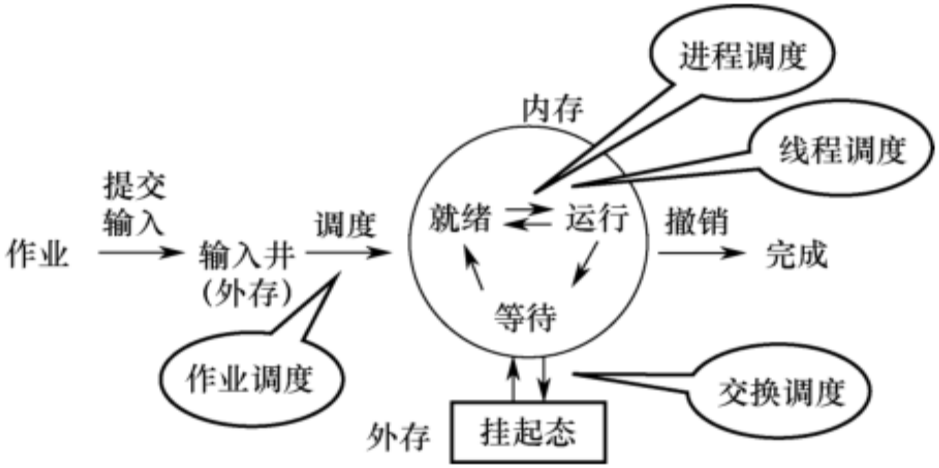


图 5-2　批处理系统的处理机调度层次

5.1.2 处理机调度的层次

1. 高级调度(High Scheduling)：

- 又称为作业调度或长程调度，主要用于批处理系统。
- 在批处理系统中，作业提交输入系统后，先驻留在外存输入井的后备队列上。高级调度负责从后备队列中选择多个作业调入内存，为它们创建进程并分配必要的资源，然后链接到就绪队列上。
- 在分时系统中，为了做到及时响应，通过键盘输入的命令或数据等，都被直接送入内存创建进程，因而不需要设置高级调度这个层次。类似地，通常实时系统也不需要高级调度。

5.1.2 处理机调度的层次

- 每当有作业执行完毕并撤离时，高级调度会选择一个或多个作业调入内存。此外，如果CPU的空闲时间超过一定的阈值，系统也会触发高级调度选择后备队列中的作业进入内存。
- 选择多少个作业进入内存取决于系统的多道程序度，选择什么类型的作业进入内存取决于系统采用的高级调度算法。
- 当进入内存的作业数目很多时，资源的利用率虽然有所提高，但每个作业的周转时间被延长了；当进入内存的作业数目太少时，虽然作业的周转时间缩短了，但系统的资源利用率和系统吞吐量又降低了。因此，系统处理程序的数量应根据系统的规模和运行速度等情况做适当的调整。

5.1.2 处理机调度的层次

2. 低级调度(Low-level Scheduling)：

- 又称为进程调度或短程调度。
- 低级调度负责按照某种调度算法，从内存的就绪队列中选择一个就绪进程将CPU分配给它。
- 低级调度是最基本的一级调度，是各类操作系统必备的功能，其调度算法的优劣将直接影响整个系统的性能。因此，这部分代码是操作系统最为核心的部分。
- 在多线程系统中，线程成为调度的基本单位，此时还存在线程调度这个层次。

5.1.2 处理机调度的层次

3. 中级调度(Intermediate-Level Scheduling)：

又称中程调度(Medium-Term Scheduling)或者交换调度。

主要目的：为了缓解内存压力，提高内存利用率。

具体实现：中级调度负责按照一定的策略，把内存中的进程交换到外存交换区，或将外存交换区中的挂起进程调入内存。

中级调度常用于分时操作系统和应用虚拟内存技术的系统中，中级调度是交换功能的一部分。

存储器管理中对换

4.同时具有三级调度的调度队列模型

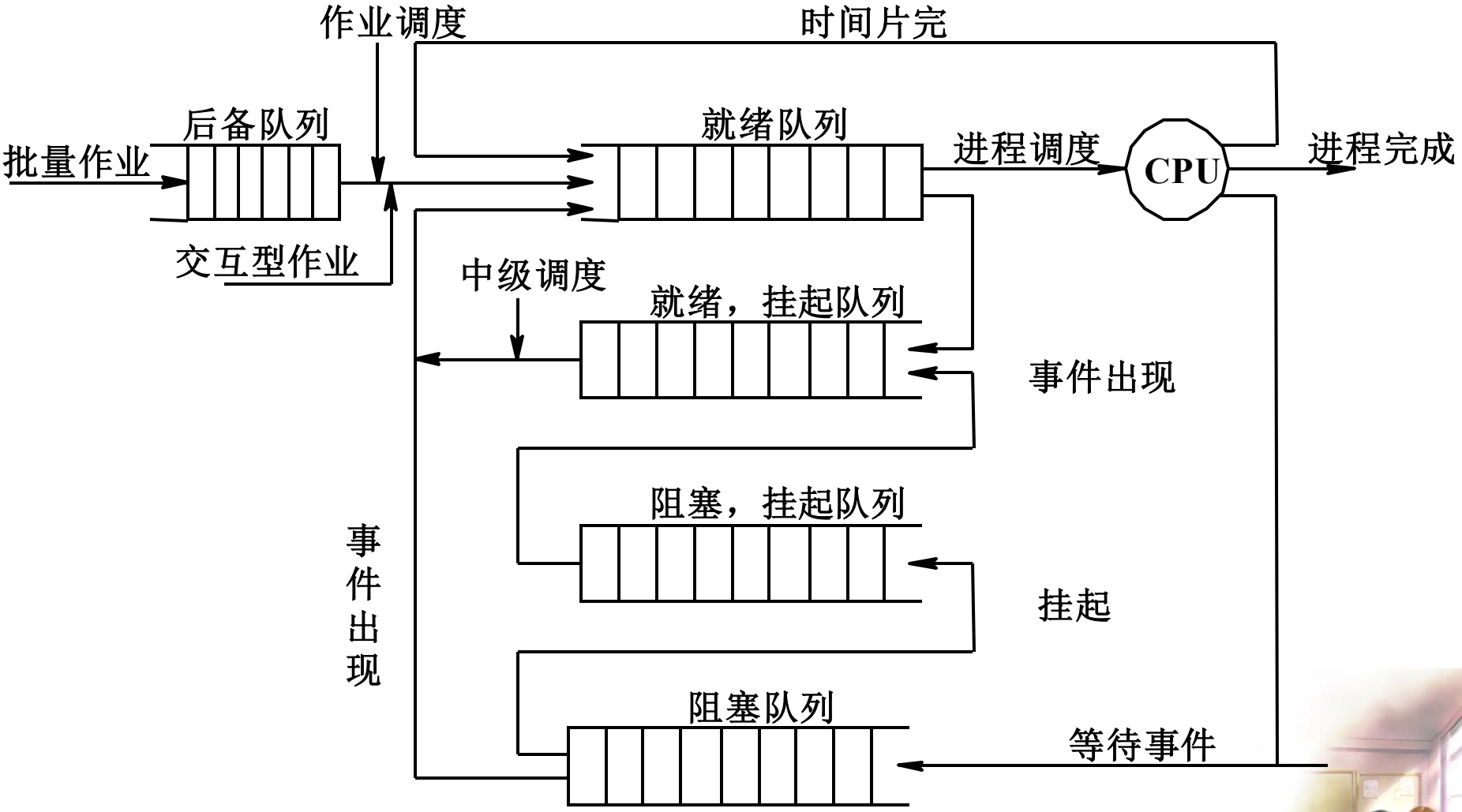


图 具有三级调度时的调度队列模型

5.1.2 处理机调度的层次

- 通常一个作业经过高级调度后，在内存会创建至少一个进程。进程在内存可能分多次被CPU 执行才能结束。在此期间可能还经历几次内存与外存的交换过程。
- 在三个调度层次中
  - 低级调度的执行频率最高，高级调度的执行频率最低，中级调度的执行频率居中。
  - 考虑到低级调度会频繁地发生，因此低级调度算法不宜太复杂。相反，高级调度的执行频率低，允许高级调度算法花费时间多一些。

5.1.3 进程调度方式与时机

- 进程调度有两种基本方式：非抢占式调度和抢占式调度。

1. 非抢占式调度，也称为非剥夺式调度。

- 采用非抢占式调度方式时，一旦把CPU分派给某个进程，该进程将一直执行下去，直至运行结束或因某种原因阻塞而不能运行，才将CPU分派给其他进程，即不允许其他进程抢占正在运行的进程的CPU。
- 在非抢占式调度方式下，只有当系统中有进程退出或进程阻塞时，才会引发进程调度，选择另外一个就绪进程投入运行。
- 优点：实现简单，系统开销小；
- 缺点：难以满足有紧急任务的进程要求。
- 分时系统和对时间要求比较严格的实时系统不使用这种方式。

5.1.3 进程调度方式与时机

2．抢占式调度，也称为剥夺式调度。

- 系统允许调度程序根据某个抢占原则，强行抢占正在运行的进程的CPU，将其分派给另外一个就绪进程。
- 抢占原则有三种：
  - 优先权原则
  - 短进程优先原则
  - 时间片原则
- 在抢占式调度方式下，进程调度的执行频率相当频繁，因此增加了进程切换的开销，但避免了任何一个进程独占CPU太长时间，可以为进程提供较好的服务。

5.1.4 调度算法选择依据与性能评价

- 处理机调度方式和调度算法的选择取决于操作系统的类型及其设计目标。
  - 批处理操作系统会选择资源利用率高、平均周转时间短和系统吞吐量大的调度算法；
  - 分时操作系统会选择交互性好、响应时间短的调度算法；
  - 实时操作系统会选择能够处理紧急任务、保证时间要求的调度算法。
- 调度算法的性能评价
  - 面向用户的性能评价准则与单个用户感知的系统行为有关，如响应时间、周转时间、优先权和截止时间保证等。
  - 面向系统的性能评价准则主要考虑系统的效率和性能，如系统吞吐率、处理机利用率和各类资源的平衡利用等。

5.1.4 调度算法选择依据与性能评价

1. 调度算法性能评价的共同准则

- 资源利用率
  - CPU 利用率成为衡量操作系统性能的重要准则。

$$\text{CPU利用率} = \frac{\text{CPU有效工作时间}}{\text{CPU有效工作时间} + \text{CPU空闲等待时间}} \times 100\%$$

- 公平性
  - 在用户或系统没有特殊要求时，进程应该被公平地对待，避免出现进程饥饿现象。

5.1.4 调度算法选择依据与性能评价

1．调度算法性能评价的共同准则（续）

- 各类资源的平衡利用

一个好的调度算法应尽可能使系统中的所有资源都处于忙碌状态，尽可能保持系统资源使用的平衡性。

- 策略强制执行

系统对所制定的策略，如抢占策略、安全策略等，必须予以准确执行，即使会造成某些工作的延迟也要执行。

- 优先级

在批处理系统、分时系统和实时系统中都可以引入优先级准则，以保证某些紧急的作业能够得到及时处理。

5.1.4 调度算法选择依据与性能评价

2. 批处理系统调度算法常用评价准则

- 周转时间
  - ① 作业在外存后备队列上等待调度的时间。
  - ② 进程在就绪队列等待调度的时间。
  - ③ 进程在CPU上的执行时间。
  - ④ 等待I/O操作完成的时间。

– 是指从作业提交给系统开始，到作业完成为止的这段时间间隔。

– 如果作业提交给系统的时刻是t1，完成的时刻是t2，那么作业的周转时间为 $T = t_2 - t_1 = T_w + T_s$

- Tw表示作业在系统中的等待时间，Ts 表示作业在系统中的运行时间。

5.1.4 调度算法选择依据与性能评价

2．批处理系统调度算法常用评价准则（续）

- 平均周转时间

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

  – 其中，Ti是第i个作业的周转时间，n是作业的个数。

- 带权周转时间

$$W=\frac{T}{T_s}=\frac{T_w+T_s}{T_s}=1+\frac{T_w}{T_s}$$

  – 用于衡量周转时间中的有效工作时间。
  – 带权周转时间总是大于1的，而且越接近1越好。

- 平均带权周转时间

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

5.1.4 调度算法选择依据与性能评价

2．批处理系统调度算法常用评价准则（续）

- 系统吞吐量
  - 显然，若处理的长作业多，则系统吞吐量低；若处理的短作业多，则系统吞吐量高。
  - 系统吞吐量是评价批处理系统性能的重要指标。

3．分时系统调度算法常用评价准则

- 响应时间
  - ① 从键盘输入的请求信息传送到处理机的时间。
  - ② 处理机对请求信息进行处理的时间。
  - ③ 将所形成的响应回送到终端显示器的时间。
  - 是指用户提交一个请求到系统响应（通常是系统有一个输出）的时间间隔。

5.1.4 调度算法选择依据与性能评价

4. 实时系统调度算法常用评价准则

- 截止时间是衡量实时系统时限性能的主要指标，也是选择实时系统调度算法的重要准则。
  - 开始截止时间：某任务必须开始执行的最迟时间
  - 完成截止时间：某任务必须完成的最迟时间。
- 可预测性
  - 例如，在视频播放任务中，视频的连续播放可以提供请求的可预测性。若系统采用双缓冲，则可以实现第i帧的播放和第i+1帧的读取并行处理，从而提高其实时性。

5.2 常用调度算法

5.2 常用调度算法

- 以表5-1所示的作业流（或进程流）为例介绍常用的调度算法。

表 5-1　作业流（或进程流）

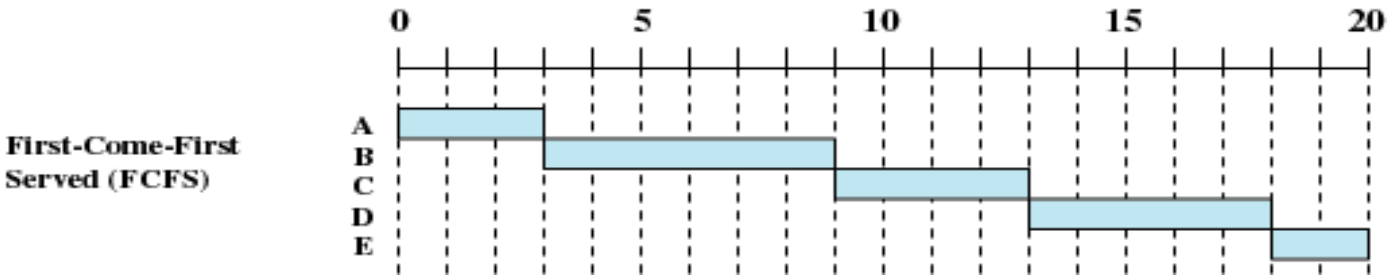
进程（作业）	到达时刻	所需服务时间/ms
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

5.2.1 先来先服务调度算法

- 先来先服务（First Come First Served，FCFS）调度算法是一种最简单的调度算法。
- 既可以用于高级调度，又可以用于低级调度。按照作业或进程到达系统的先后次序进行调度。
  - 用于高级调度时，每次从后备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，然后将进程链接到就绪队列。
  - 用于低级调度时，每次从就绪队列中选择一个最先就绪的进程，把CPU分派给它，使之投入运行，一直到该进程运行完毕或阻塞后，才让出CPU。
- FCFS调度算法是一种非抢占式调度算法。

First-Come-First-Served (FCFS)

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	3	0	3	3	1
B	2	6	3	9	7	1.17
C	4	4	9	13	9	2.25
D	6	5	13	18	12	2.4
E	8	2	18	20	12	6
					8.60	2.56



A、B、C、D四个进程分别到达系统的时间、要求服务的时间，写出开始执行时间及完成时间并计算出各自的周转时间和带权周转时间。

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*
					100	*26*

短进程C的带权周转时间高达100，而长进程D的带权周转时间仅为1.99。短进程C为了等待长进程B执行结束，其周转时间和带权周转时间将变得很长，从而使平均周转时间和平均带权周转时间也变得很长。

5.2.1 先来先服务调度算法

- 优点：
  - FCFS调度算法简单、易于实现。
  - 有利于长进程。
  - 有利于CPU繁忙的进程。
- 缺点：
  - 不利于短进程。FCFS调度算法只考虑了进程等待时间的长短，未考虑进程要求服务时间的长短，不利于短进程，尤其对于短进程紧随长进程的情况更不适用。
  - 不利于I/O操作繁忙的进程。对于I/O操作繁忙的进程，每进行一次I/O操作都要等待系统中其他进程一个运行周期结束后才能再次获得CPU，故大大延长了进程运行的总时间，也不能有效利用各种外设资源。

5.2.2 短进程（作业）优先调度算法

适合于作业调度和进程调度

- **Shortest Process First(SPF)或 Shortest Job First(SJF)**
- 该算法优先选择短进程（作业）投入运行，以非抢占式为例。
  - **SJF调度算法是从后备队列中选择一个或多个估计运行时间最短的作业，将它们调入内存运行。**
  - **SPF调度算法是从就绪队列中选择一个估计运行时间最短的就绪进程，将CPU分派给它，使其执行。**

例1 FCFS和SPN调度算法的性能比较

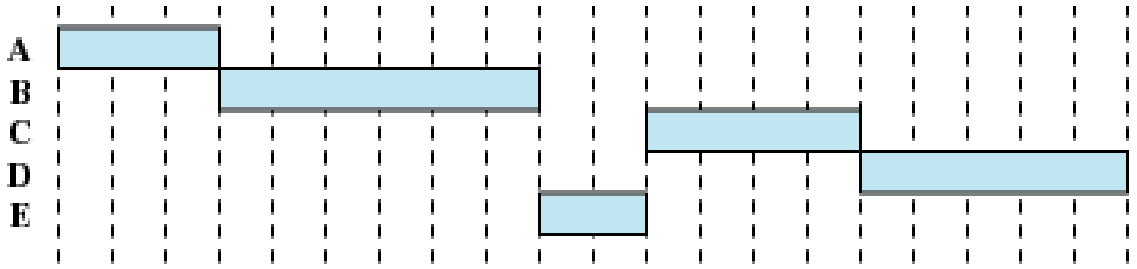
FCFS

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	3	0	3	3	1
B	2	6	3	9	7	1.17
C	4	4	9	13	9	2.25
D	6	5	13	18	12	2.4
E	8	2	18	20	12	6
					8. 60	2.56

SPN

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	3	0	3	3	1
B	2	6	3	9	7	1.17
C	4	4	11	15	11	2.75
D	6	5	15	20	14	2.8
E	8	2	9	11	3	1.5
					7.60	1.84

Shortest Process Next (SPN)



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

FCFS和SJF调度算法的性能比较

5.2.2 短进程（作业）优先调度算法

优点:(1)能有效降低作业的平均等待时间。

(2)提高系统的吞吐量。

缺点:(1)不利于长进程(作业)，尤其是在系统不断地有短进程(作业)到达的情况下，会导致长进程(作业)的饥饿现象。

(2)没有考虑进程(作业)的紧迫程度，不利于处理紧急任务，对于分时、实时操作处理仍然不理想。

(3)必须预知进程(作业)的运行时间。由于进程(作业)运行时间的长短只是根据用户所提供的估计执行时间而定的，而用户又可能会有意或无意地缩短其作业的估计运行时间，致使该算法不一定能真正做到短进程(作业)优先调度，从而影响调度性能。

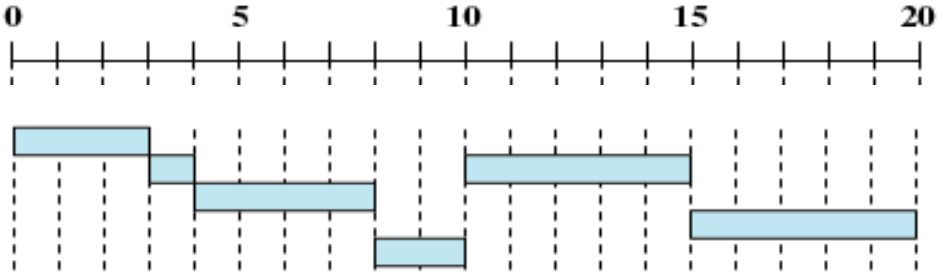
补充内容：最短剩余时间优先调度算法 Shortest Remaining Time

- 简称SRT。
- 调度时选择预期剩余时间最短的进程。
- 当一个新进程加入到就绪队列时，它可能比当前运行的进程具有更短的剩余时间。因此，只要新进程就绪，调度器可能抢占当前正在运行的进程。
- 也可能存在长进程被饿死的危险。

最短剩余时间

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Shortest Remaining Time (SRT)



5.2.3 轮转调度算法

- 基于时钟的轮转（Round Robin，RR）调度算法用于低级调度，专为分时操作系统设计。
  - 假设系统有n个就绪进程，RR调度算法是将CPU的处理时间划分成n个大小相等的时间片，系统将所有的就绪进程按先来先服务原则排成一个就绪队列，每次调度时将就绪队列的队首进程分派到CPU上执行，并令其只能执行一个时间片；当时间片用完时，调度程序中止当前进程的执行，将它送到就绪队列的队尾，再调度下一个队首进程执行。也就是说，RR调度算法是以时间片为单位轮流为每个就绪进程服务的，从而保证所有的就绪进程在一个确定的时间段内，都能够获得一次CPU执行。
- RR调度算法是一种抢占式调度算法。

时间片大小的确定

退化成

- 时间片太大 ⟶ FCFS算法
- 时间片过小切换开销大。
- 因此，时间片的长度要略大于一次典型交互活动所需的时间，通常为10~100ms 。

时间片大小确定要考虑的因素：

(1) 系统对响应时间的要求。（用户数一定时，成正比）

(2)就绪队列中的进程数目。（保证响应时间，成反比）

(3)系统的处理能力。（保证用户键入的命令能在一个时间片内处理完毕）

5.2.3 轮转调度算法

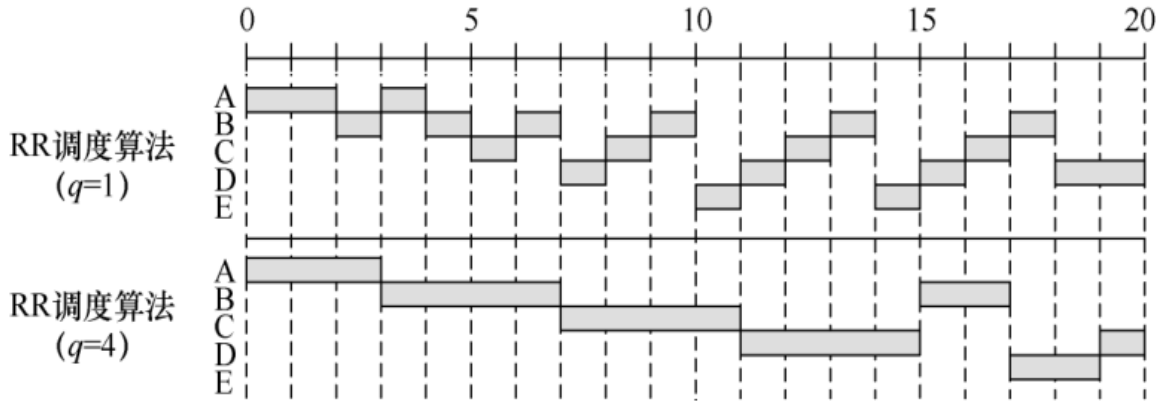


表 5-5　RR 调度算法性能

时间片	进程	到达时刻/ms	服务时间/ms	开始执行时间/ms	完成时间/ms	周转时间/ms	带权周转时间
$q=1$	A	0	3	0	4	4	1.33
	B	2	6	2	18	16	2.67
	C	4	4	5	17	13	3.25
	D	6	5	7	20	14	2.8
	E	8	2	10	15	7	3.5
	平均值	—	—	—	—	10.8	2.71
$q=4$	A	0	3	0	3	3	1
	B	2	6	3	17	15	2.5
	C	4	4	7	11	7	1.75
	D	6	5	11	20	14	2.8
	E	8	2	17	19	11	5.5
	平均值	—	—	—	—	10	2.71

5.2.3 轮转调度算法

- 特点：
  - RR调度算法简单易行，进程的平均响应时间短，交互性好，但不利于处理紧急任务。因此，RR 调度算法适用于分时操作系统，但不适用于实时系统。
  - 该算法不利于处理I/O 操作频繁的进程，因为这些进程通常运行不完一个时间片就阻塞了，等它完成了I/O 操作后，又要和其他进程一样排队。所以这类进程的周转时间会比不需要I/O 操作或I/O 操作少的进程要长得多。

改进的RR 调度算法

- 新进程到达后，先进入就绪队列。调度时，按照先来先服务原则，以时间片为单位进行调度。
  - 该进程的时间片用完后，仍然转入就绪队列队尾排队。
  - 当该进程因I/O 操作而被阻塞时，会转入某个阻塞队列。当I/O 操作完成，该阻塞进程被唤醒时不是进入就绪队列而是进入一个辅助队列。
- 在调度时，辅助队列的进程优于就绪队列的进程。
- 改进的RR 调度算法在公平性方面优于一般的RR调度算法。

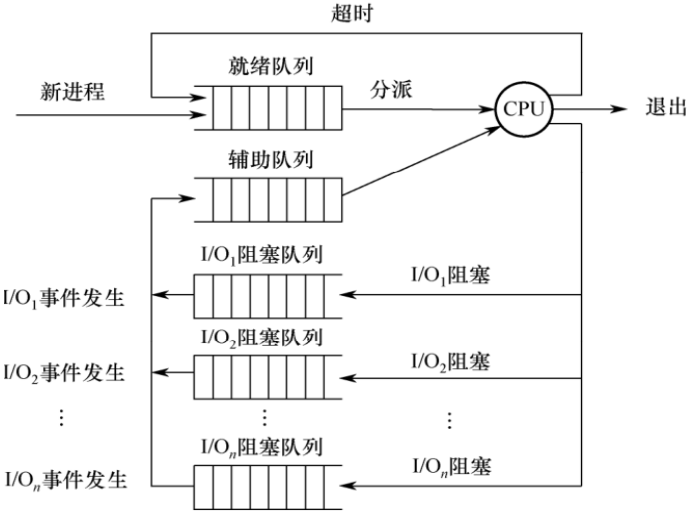


图 5-4　改进的 RR 调度算法

5.2.4 优先级调度算法

适用于作业调度和进程调度

- 调度程序总是选择优先级最高的就绪进程，分派占用CPU。
- 优先级相同的进程，则按照先来先服务原则进行调度。SPF 调度算法就是优先级调度算法的一个特例，进程的优先级依赖于进程的长度。

5.2.4 优先级调度算法

1. 优先级调度算法的类型

(1)非抢占式优先级算法（常用于批处理、要求不严的实时）

系统一旦把处理机分配给就绪队列中优先级最高的进程后，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级最高的进程。

(2) 抢占式优先级调度算法（常用于要求严格的实时、性能要求较高的批处理和分时）

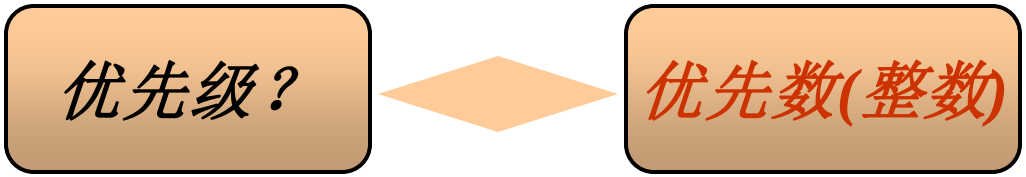
系统把处理机分配给优先级最高的进程，使之执行。但在其执行期间，只要又出现了另一个优先级更高的进程，进程调度程序就立即停止当前进程(原优先级最高的进程)的执行，重新将处理机分配给新到的优先级最高的进程。

每当一个进程就绪后，系统都要按照优先级抢占原则判断是否进行抢占，导致调度开销大为增加。但这种抢占式的优先级调度算法，能更好地满足紧迫作业的要求。

2. 优先级的类型

1) 静态优先级

静态优先级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。



确定进程优先级的依据有如下三个方面：

(1) 进程类型

系统进程高，一般用户进程低。

(2) 进程对资源的需求

进程的估计执行时间、内存需要量等。

(3) 用户要求

紧迫程度、所付费用。

静态优先级法的优缺点：

优点：简单易行、系统开销小。

缺点：灵活性较差，可能出现低优先级作业或进程长期得不到调度（饥饿）的情况。

2) 动态优先级

进程的优先级可以随进程的推进或随其等待时间的增加而改变，以便获得更好的调度性能。

例如，在就绪队列中的进程，随其等待时间的增长，其优先级以速率a提高。若所有进程都具有相同的优先级初值，则FCFS算法。若所有的就绪进程具有各不相同的优先级初值，那么对初值低的进程，在等待了足够时间后，其优先级便可能升为最高。当采用抢占式优先级调度算法时，如果再规定当前进程的优先级以速率b下降，则可防止一个长作业长期地垄断处理机。

5.2.5 最高响应比优先调度算法

（动态优先级机制）

非抢占式调度算法

优先级的变化规律可描述为：

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

由于等待时间与服务时间之和，就是系统对该作业的响应时间，故该优先级又相当于响应比R_P。据此，又可表示为：

$$\text{响应比} = \frac{\text{等待时间+要求服务时间}}{\text{要求服务时间}} = \frac{\text{响应时间}}{\text{要求服务时间}}$$

HRRN

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	3	0	3	3	1
B	2	6	3	9	7	1.17
C	4	4	9	13	9	2.25
D	6	5	15	20	14	2.8
E	8	2	13	15	7	3.5
					8.0	2.14

- Rc=(9-4+4)/4=2.25　　Rd=(9-6+5)/5=1.6

  Re=(9-8+2)/2=1.5
  - 调度C执行。
- Rd=(13-6+5)/5=2.4　　Re=(13-8+2)/2=3.5
- 调度E执行。

➢ 通过对比可以看出，HRRN调度算法的平均带权周转时间介于FCFS调度算法和SPF调度算法之间。

算法分析：

(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，避免了饥饿现象。

因此，HRRN调度算法既照顾了短进程（作业），又不会使长进程（作业）的等待时间过长，有效提高了调度的公平性。

缺点：每次进行调度之前，都需要先做响应比的计算，会增加系统开销。

5.2.6 多级队列调度算法

- 上述的几种调度算法通常只设置一个就绪队列，采用的进程调度算法也是单一的。如果系统需要根据进程的类型采用不同的进程调度算法，可以考虑采用多级队列调度算法。
- 多级队列调度算法的主要思想：
  - 组建多个就绪队列，进程就绪后根据其类型或性质链接到相应的就绪队列，不同的就绪队列可以设置不同的优先级，同一个就绪队列中的进程也可以设置不同的优先级。
  - 由于设置多个就绪队列，允许对每个就绪队列实施不同的调度算法，以满足不同用户进程的需求，实现调度策略的多样性。
  - 多处理机系统中，该算法可以很方便地为每个处理机设置一个单独的就绪队列。

5.2.7 多级反馈队列调度算法

- 如果没有各个进程相对长度的信息，SPF 调度算法、HRRN 调度算法等基于进程长度的调度算法都不能使用。
- 既然无法获得要求服务时间的信息，人们就考虑利用进程的已执行时间来进行调度。
- 多级反馈（Feedback，FB）队列调度算法就不必事先知道各进程所需的执行时间，而且还可以满足各种类型进程的需要，它是目前被公认的一种较好的进程调度算法。
- FB 调度算法被很多操作系统所采用，最典型的有 Windows NT 和UNIX。

5.2.7 多级反馈队列调度算法

（1）系统设置多个就绪队列，每个就绪队列有不同的优先级。第1级就绪队列的优先级最高，以下各级就绪队列的优先级逐次降低。优先级高的队列会优先调度执行。同一就绪队列的进程优先级相同，按照先来先服务原则调度。

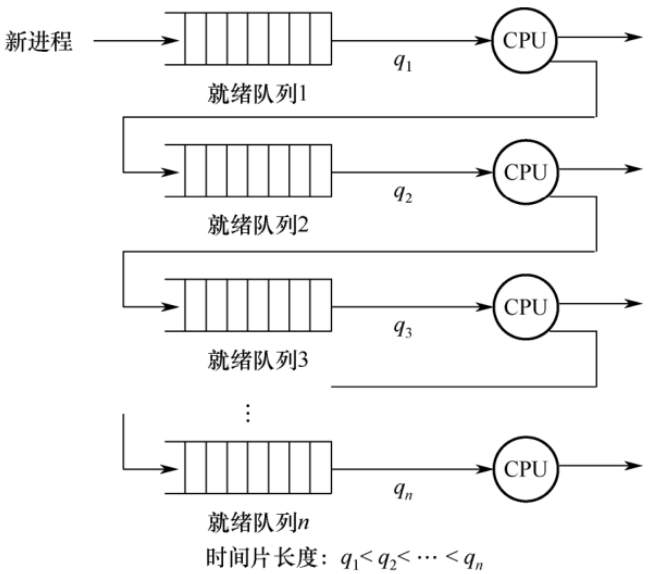


图 5-5　FB 调度算法

（2）为每个队列设置大小不同的时间片。为优先级最高的第1级就绪队列中的进程设置的时间片最小，随着队列的级别增加，其进程的优先级逐级降低，但被赋予的时间片逐级增加。

例如，$q = 2^{i-1}$，其中i为队列编号。

5.2.7 多级反馈队列调度算法

（3）新进程先进入第一个就绪队列，并按先来先服务原则等待调度。当调度到该进程时，该进程若能在一个时间片内完成，便可退出；若它在规定的时间片内未能完成，就转入下一级队列末尾等待调度，依此类推。在第n个队列中便采用时间片轮转的方式运行。

（4）系统总是从第一个就绪队列的进程开始调度；只有前i-1 级就绪队列都为空时，才调度第i 级就绪队列中的进程。当一个进程在运行时，更高优先级的就绪队列中来了一个进程，那么高优先级的进程可以抢占当前运行进程的CPU。被抢占的进程可以排到原就绪队列末尾，也可仍留在队首，以便下次重新获得CPU时把原先分配到的时间片的剩余部分用完。

5.2.7 多级反馈队列调度算法

- **FB** 调度算法是一种具有动态优先级机制的调度算法。
  - 它根据进程运行情况的反馈信息，能够动态地调整不同类型的进程在不同运行阶段的优先级，重新调整所处队列，因此具有自适应的能力。

- 多级反馈队列调度算法的性能

(1) 终端型作业用户。

交互型作业，通常较小，第一队列一个时间片即可完成

(2) 短批处理作业用户。

第一队列一个时间片即可完成，或第一队列、第二队列各一个时间片

(3) 长批处理作业用户。

可能到第N个队列，按时间片轮转，不必担心得不到处理

FB 调度算法贯彻了处理机调度策略中“要得越多，等待的时间也应越长”的原则，能较好地满足各种类型用户的需要。

5.2.7 多级反馈队列调度算法

- 当然，在FB 调度算法下，进程仍然存在饥饿的可能。
  - 因为一个长进程会很快沉底，位于优先级最低的就绪队列中。不断到来的短进程如果形成稳定的进程流，长进程就会永远等待下去。
- 一个改进的方法是：
  - 记录一个进程在某个队列中已经等待的时长。若这个时长超出了允许范围，则将该进程提升到上一级就绪队列。
  - 这样，获得CPU 的可能性也随之增加。
- 在实际应用中，FB 调度算法还有很多的变体。
  - 例如，为了保证I/O 操作能及时完成，可以在进程发出I/O 请求后进入最高优先级队列，并执行一个时间片来响应I/O 操作。

5.3 实时系统的调度

5.3 实时系统的调度

- 实时系统的进程或任务往往带有某种程度的紧迫性。计算机必须在一定的时间内对它们做出正确的反应。
  - 例如，医院里的重病监护系统、飞行器的自动导航和核反应堆的安全控制等。
- 实时操作系统具有以下特点：
  - ①有限等待时间（决定性）；
  - ②有限响应时间；
  - ③用户控制；
  - ④可靠性高；
  - ⑤系统处理能力强。

5.3.1 实时调度实现要求

1. 实时任务的类型

- 根据处理时限的要求不同，实时任务可以分为：
  - 硬实时任务：要求系统必须完全满足任务的时限要求。
  - 软实时任务：允许系统对任务的时限要求有一定的延迟，其时限要求只是一个相对条件。
- 根据外部事件的发生频率不同，实时任务还可以分为：
  - 周期任务和非周期任务。

5.3.1 实时调度实现要求

- 当系统处理的是多种周期任务时，计算机能否及时处理所有的事件取决于事件的到达周期和需要处理的时间。
  - 假设系统有m个周期事件，如果事件i 的到达周期为Pi，所需CPU的处理时间为Ci，那么只有满足下面的限制条件：

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

  系统才是可调度的。

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10ms，不难算出，此时系统是不可调度的。

解决方法是提高系统的处理能力。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，显著地减少每个任务的处理时间；其二是采用多处理机系统。假定系统中处理机数为N，则应将限制条件改为：

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

5.3.1 实时调度实现要求

2. 实时调度的实现要求

- 具有快速的切换机制
  - 对中断的快速响应能力、快速的任务分派能力。
- 采用基于优先级的抢占式调度策略
  - 基于优先级的固定点抢占式调度算法、基于优先级的随时抢占式调度算法
- 系统处理能力强

5.3.2 实时调度算法

- 实时任务都有截止时间的要求：
  - 开始截止时间、完成截止时间。

1. 保证开始截止时间的实时调度算法

- 最早截止时间优先（Earliest Deadline First，EDF）调度算法是广泛使用的一种保证开始截止时间的实时调度算法。
  - EDF调度算法根据任务的开始截止时间来确定任务的优先级。
  - 任务的开始截止时间越早，其优先级越高。具有最早开始截止时间的任务排在就绪队列队首，被优先调度执行。

(1) 非抢占式调度方式用于非周期实时任务

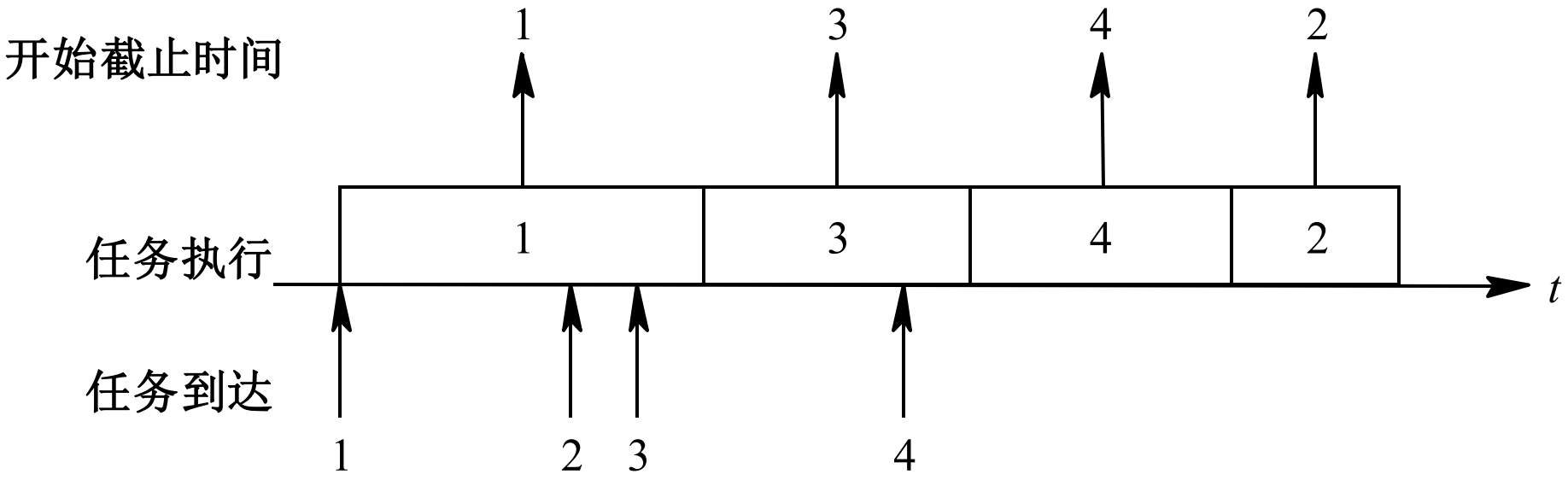


图 EDF算法用于非抢占调度方式

(2) 抢占式调度方式用于周期实时任务

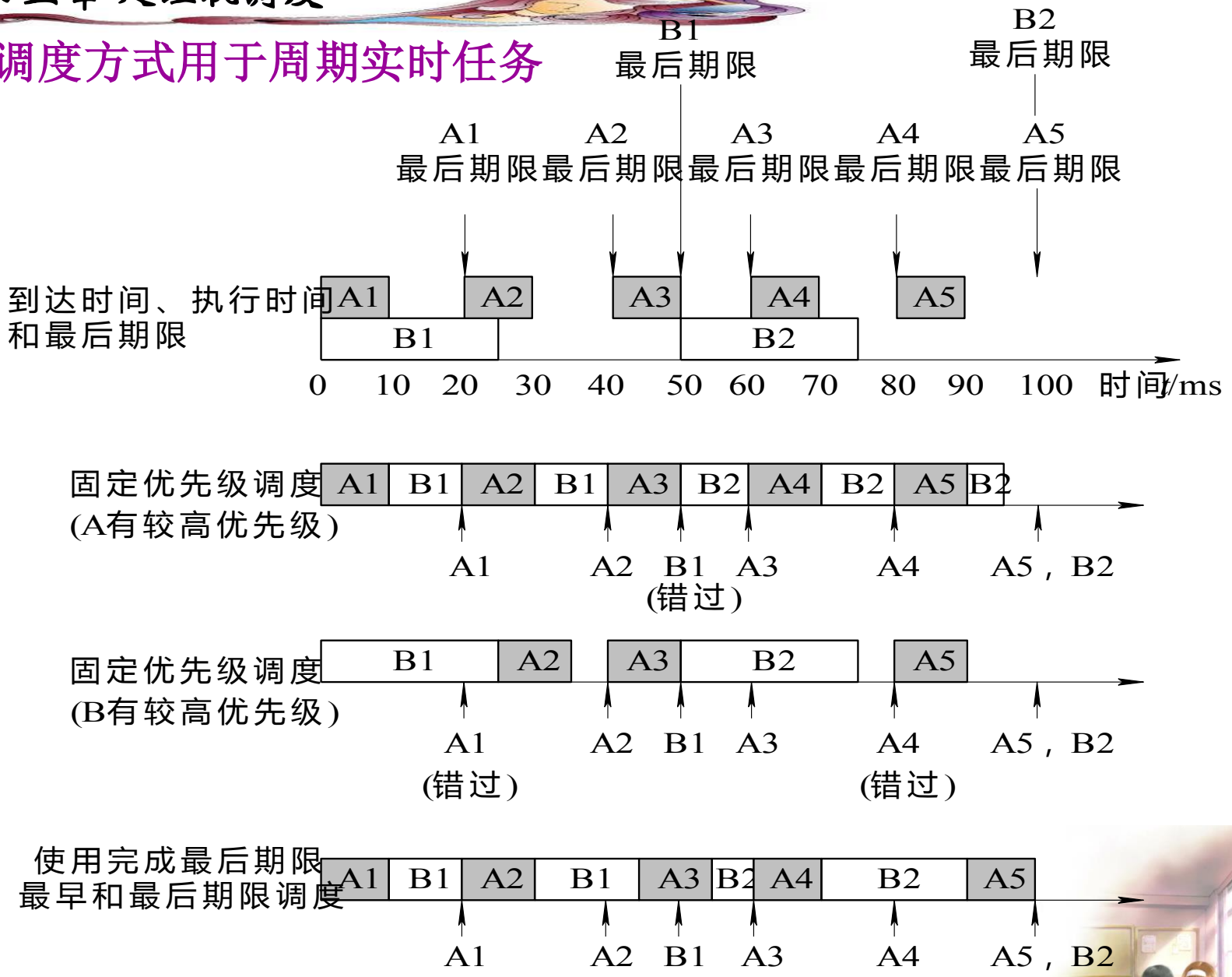


图 最早截止时间优先算法用于抢占调度方式之例

5.3.2 实时调度算法

2．保证完成截止时间的实时调度算法

- 最低松弛度优先（Least Laxity First，LLF）调度算法是一种保证完成截止时间的实时调度算法。
- LLF调度算法根据任务紧急或松弛程度确定任务的优先级。
  - 任务的松弛度可定义为：

  任务的松弛度=完成截止时间−还需运行的时间−当前时间

  - 任务的松弛度越低（即紧急程度越高），其优先级越高，越优先调度执行。

最低松弛度优先LLF(Least Laxity First)算法

该算法主要用于可抢占调度方式中。

假如在一个实时系统中，有两个周期性实时任务A和B，任务A要求每20ms执行一次，执行时间为 10ms；任务B只要求每50ms执行一次，执行时间为25ms。

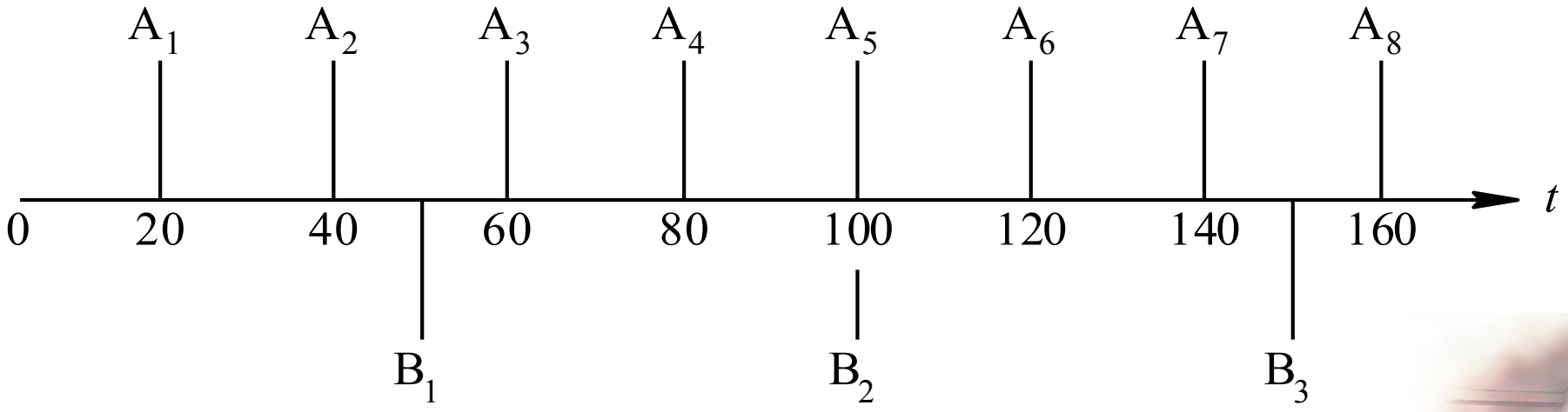


图　A和B任务每次必须完成的时间

在刚开始时(t_1=0)，A_1必须在20ms时完成，而它本身运行又需10ms，可算出A_1的松弛度为10ms；B_1必须在50ms时完成，而它本身运行就需25ms，可算出B_1的松弛度为25 ms，故调度程序应先调度A_1执行。在t_2=10ms时，A_2的松弛度可按下式算出：A_2离开始截止时间（松弛度）=必须完成时间-其本身的运行时间-当前时间

=40 ms-10 ms-10 ms=20 ms

类似地，可算出B_1的松弛度为15ms，故调度程序应选择B_1运行。在t_3=30ms时，A_2的松弛度已减为0(即40-10-30)，而B_1的松弛度为15ms(即50-5-30)，于是调度程序应抢占B_1的处理机而调度A_2运行。在t_4=40ms时，A_3的松弛度为10ms(即60-10-40)，而B_1的松弛度仅为5ms(即50-5-40)，故又应重新调度B_1执行。在t_5=45ms时，B_1执行完成，而此时A_3的松弛度已减为5ms(即60-10-45)，而B_2的松弛度为30ms(即100-25-45)，于是又应调度A_3执行。在t6=55ms时，任务A尚未进入第4周期，而任务B已进入第2周期，故再调度B2执行。在t7=70ms时，A4的松弛度已减至0ms(即80-10-70)，而B2的松弛度为20ms(即100-10-70)，故此时调度又应抢占B2的处理机而调度A4执行。

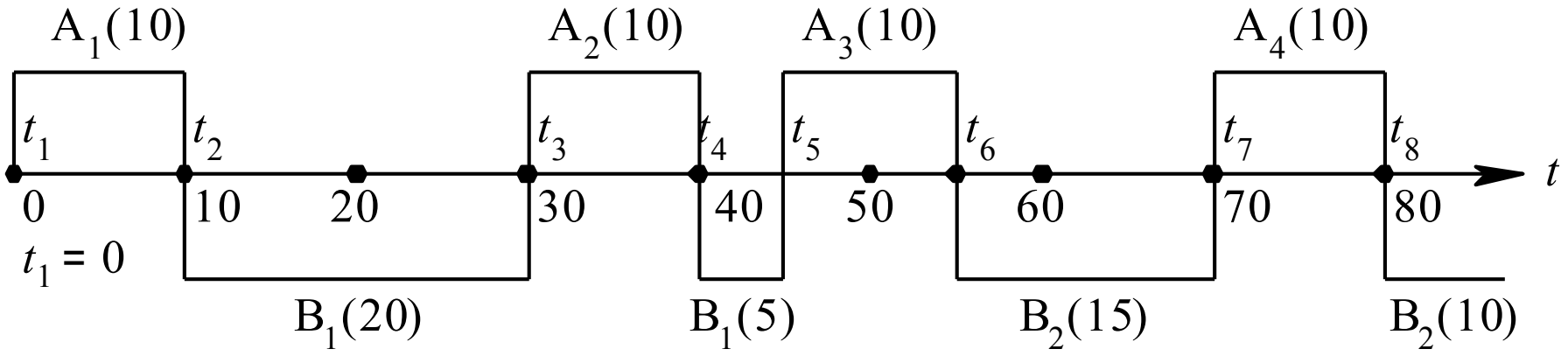


图　利用ELLF算法进行调度的情况

5.3.3 优先级倒置

- 发生在基于优先级的抢占式调度策略下的一种现象。

1. 优先级倒置的形成

- 优先级倒置是指系统出现了高优先级的进程被低优先级的进程延迟或阻塞，对实时任务的及时完成造成很大破坏的现象。
- 系统有进程A、进程B和进程C，优先级依次从低到高。进程A和进程C共享同一个临界资源R，P(mutex)是实现临界资源R互斥访问的信号量。在t3时刻，高优先级的进程C被低优先级的进程A阻塞，出现了优先级倒置的现象。

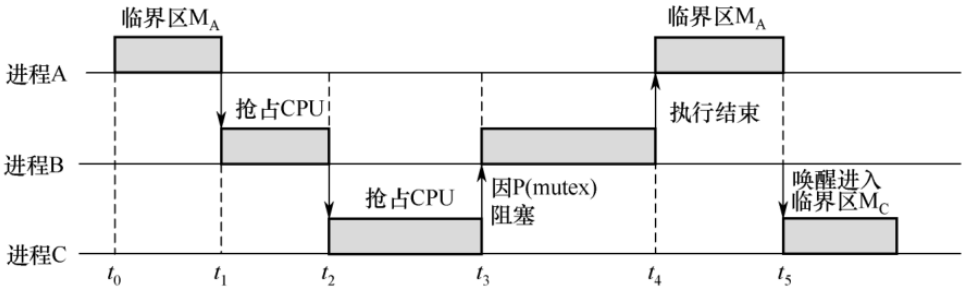


图 5-8　优先级倒置的示例图

5.3.3 优先级倒置

2. 优先级倒置的解决方案
   - 规定进程进入临界区后，不允许被更高优先级的进程抢占CPU，从而可以让该进程尽快执行完临界区，避免阻塞竞争同一临界资源的更高优先级的进程。
   - 采用动态优先级继承策略。规定某进程进入临界区后，若存在竞争同一临界资源的更高优先级的进程也要进入临界区，则阻塞更高优先级的进程，并把更高优先级转移到正占有临界资源的进程作为其优先级。

5.3.3 优先级倒置

- 在图5-8中，在t3时刻，进程C因执行P(mutex)获得不了临界资源而阻塞，此时进程A正占有该临界资源，则进程A继承进程C的优先级。
- 在t3时刻，因为进程A的优先级高于进程B的优先级，系统调度进程A执行，从而可使进程A尽快执行完临界区。如图5-9，进程A执行完临界区，会唤醒进程C，同时进程C和进程A的优先级恢复原来的初始设置，维护了进程本来的紧要程度。

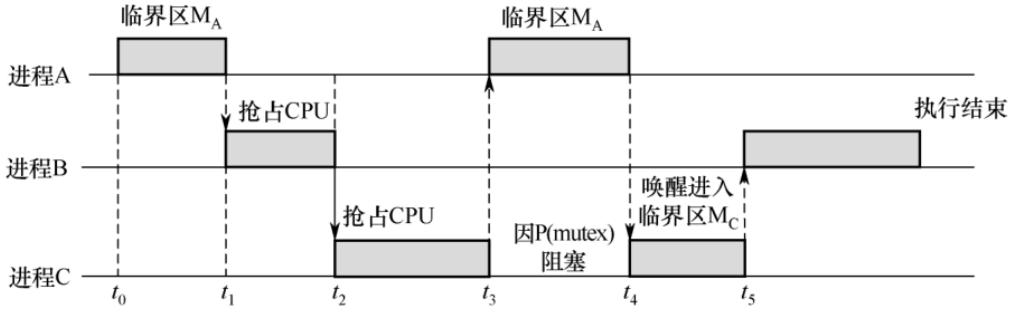


图 5-9　采用动态优先级继承策略解决优先级倒置问题

第五章　小结

- 处理机调度的层次
- 处理机调度方式
- 调度算法的目标
- 重点理解各种调度算法，比较优缺点
- 实时调度
- 优先级倒置