第三章 处理器调度

课程结构

操作系统原理

- 背景
  - Chapter 1 计算机系统
  - Chapter 2 操作系统
- 进程
  - Chapter 3 进程控制和管理
  - Chapter 5 并发：互斥与同步
  - Chapter 6 并发：死锁
- 内存
  - Chapter 7 存储管理
  - Chapter 8 虚拟内存
- 调度
  - Chapter 4 处理器调度
- 输入输出（I/O）
  - Chapter 9 设备管理
- 文件
  - Chapter 10 文件系统

本章教学目标

- 理解处理器调度的三个层次
- 掌握处理器调度算法及其评价方法

处理器调度的目的

- 作业数量众多，而处理器、内存资源有限
- 将处理器分配给不同的进程，以提高
  - 响应时间
  - 吞吐能力
  - 处理器效率

3.1 调度的层次

- 长程调度(long-term scheduling)
  - 又称高级调度、作业调度
  - 决定了哪些作业被允许进入系统参与CPU的竞争
  - **高级调度将控制多道程序的道数**
- 中程调度(medium-term scheduling)
  - 又称中级调度、平衡调度
  - 根据主存状态决定主存中所能容纳的进程数目。当主存资源紧缺时，决定将哪些进程交换出内存；而当主存资源空闲时，选择将哪些进程交换回内存。
- 短程调度(short-term scheduling)
  - 又称低级调度、进程调度/线程调度、CPU调度
  - 决定将就绪队列中的哪个进程/内核级线程分配处理器资源，使其能占用CPU执行。
  - 低级调度是操作系统最核心的部分，执行频繁

3.1 调度的层次

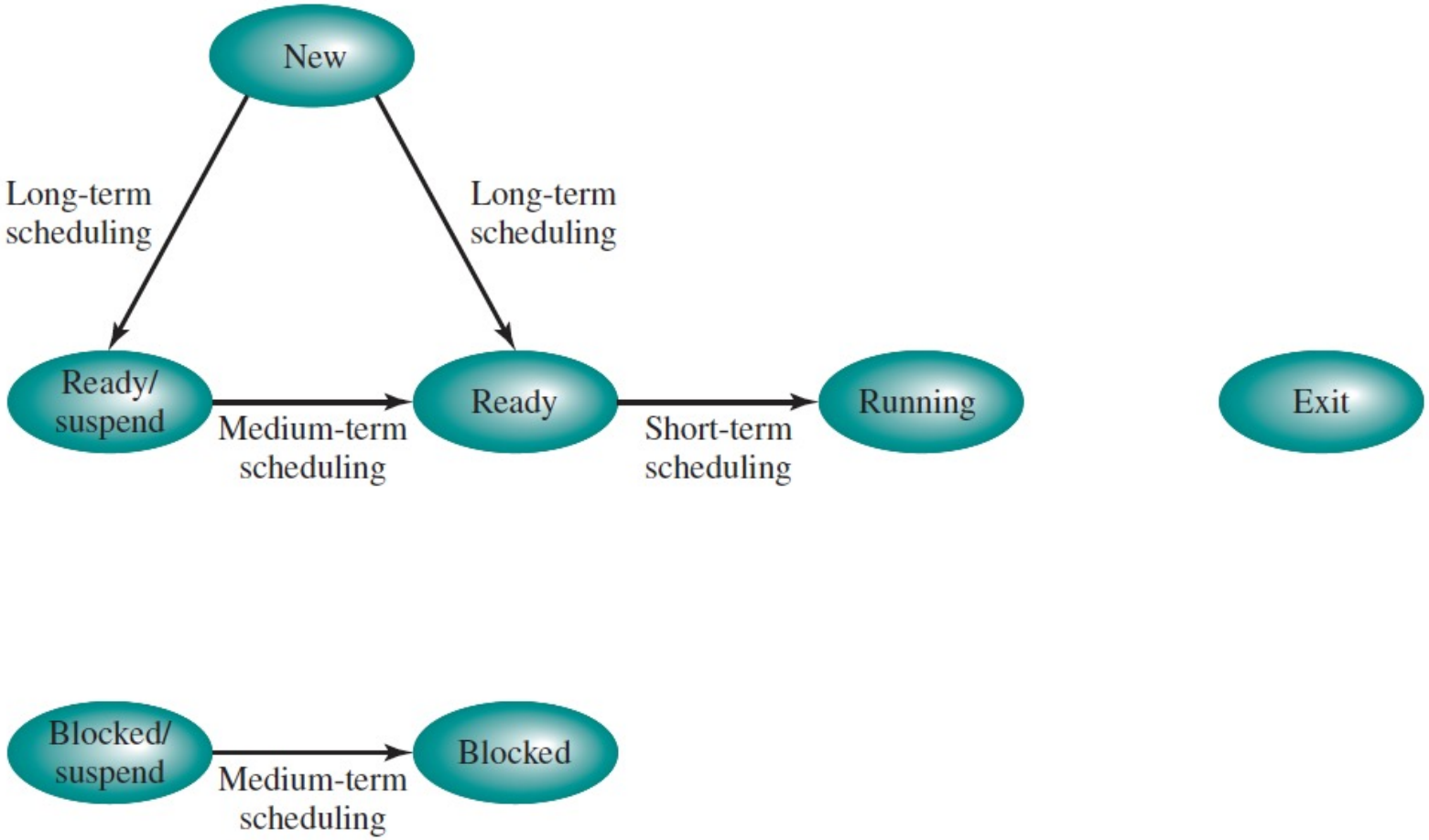


Figure 9.1 Scheduling and Process State Transitions

3.1 调度的层次

调度层次与进程状态的关系图

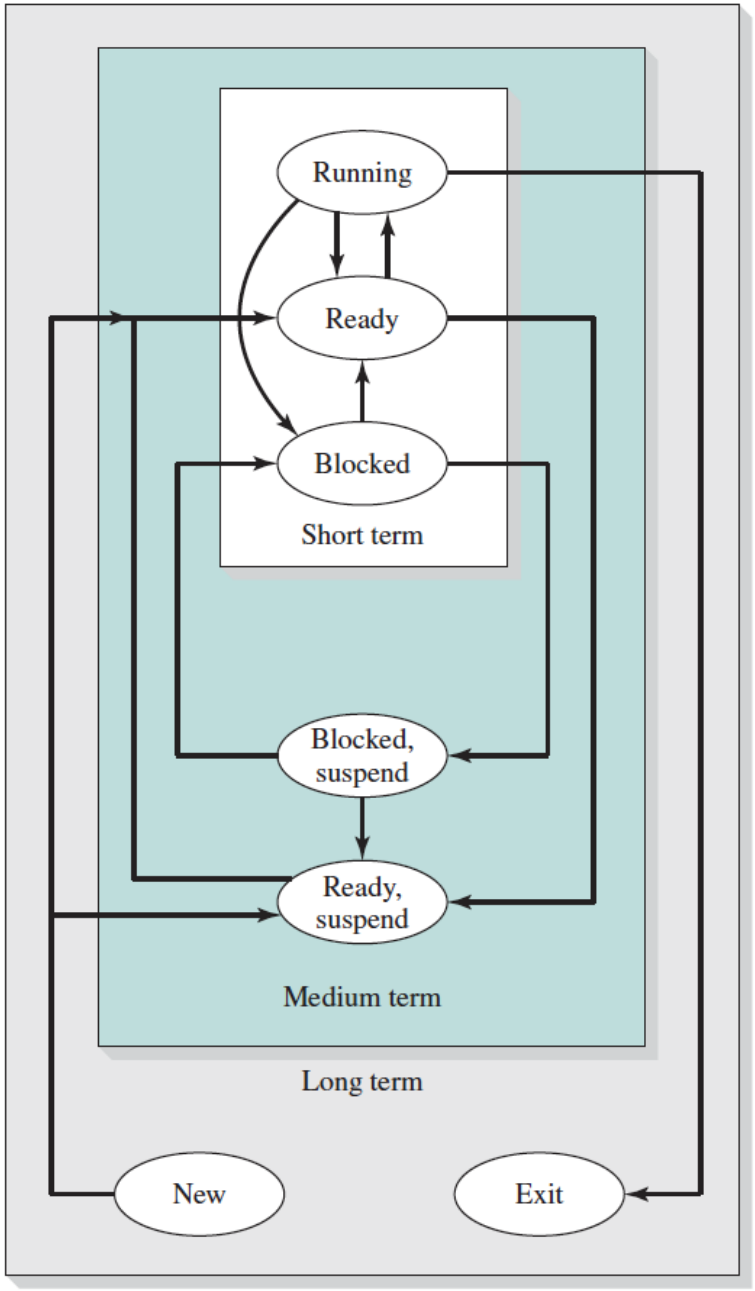


Figure 9.2 Levels of Scheduling

3.1 调度的层次

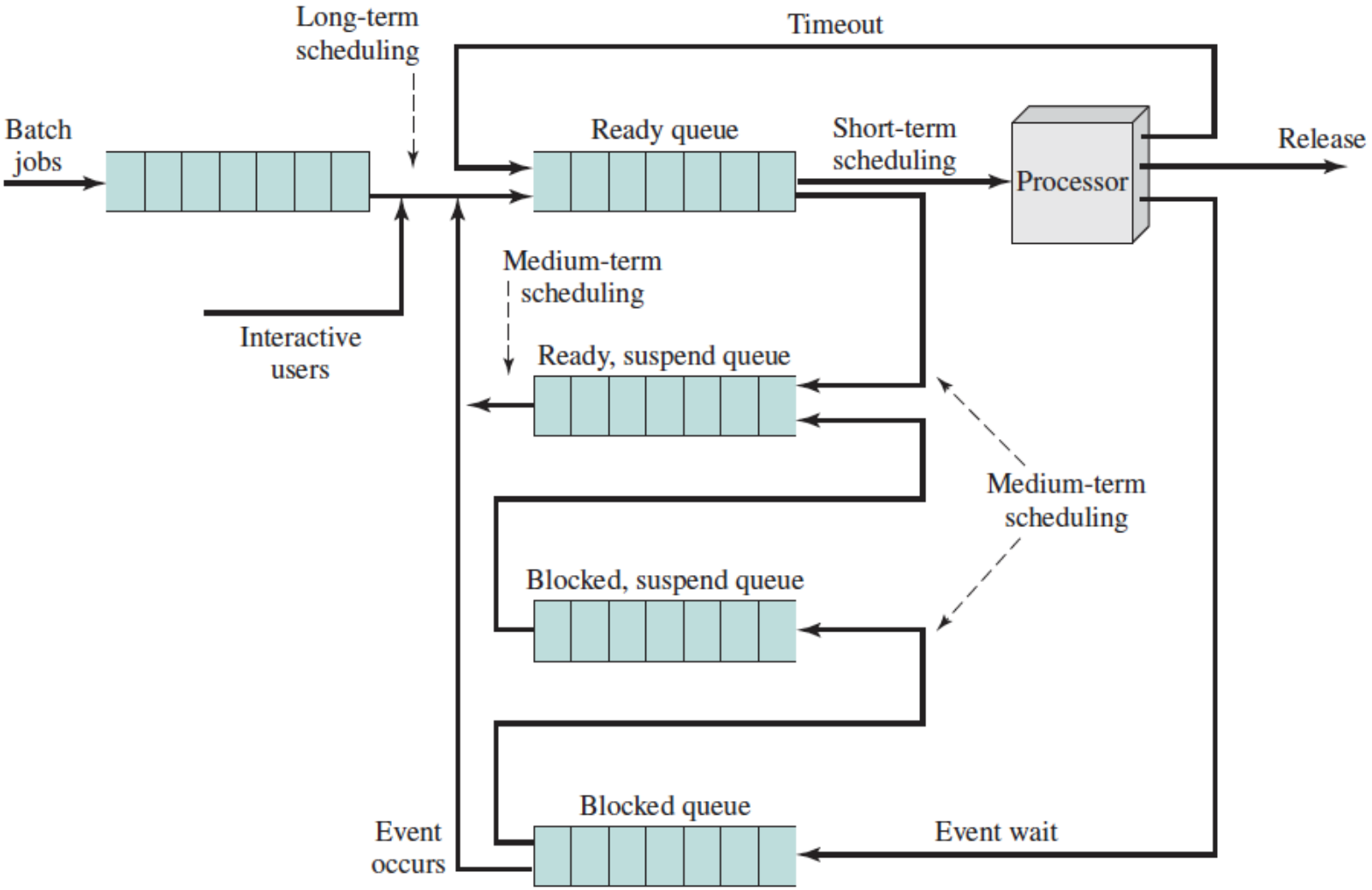


Figure 9.3 Queueing Diagram for Scheduling

3.1 调度的层次

- 批处理系统
  - 何时从后备作业队列中创建新进程？
    - 一个作业运行结束
    - CPU的空闲时间比例超过某个门限值
  - 选择哪些作业进入主存，使其成为进程？
    - FCFS, SJF, ...
    - 平衡CPU密集型作业和I/O密集型作业
    - 平衡I/O资源的使用
- 时分系统
  - 所有授权用户都被准入，直至系统饱和

3.1 调度的层次

- 又称分派器，执行最为频繁
- 短程调度的执行时刻
  - 时钟中断
  - I/O中断
  - 系统调用
  - 信号

3.2 调度算法

- 调度算法的评价指标
- 调度策略

3.2 调度算法—评价指标

- 吞吐率
  - 单位时间完成的进程数
- 响应时间
  - 从请求提交到开始接收到响应的时间，适用于交互型进程
- 处理器利用率
  - CPU利用率=CPU有效工作时间/（CPU有效工作时间+CPU空闲等待时间）
- 周转时间
  - 进程从提交到结束的时间，包括执行时间和等待时间，适用于批处理作业
- 截止时间
  - 进程必须在给定截止时间前完成，适用于实时任务
- 公平性
  - 确保每个进程过的合理的CPU份额和其他资源分配，避免出现进程饿死。

3.2 调度算法—评价指标

- 平均作业周转时间

$$T = (\sum_{i=1}^{n} t_i) / n = (\sum_{i=1}^{n} (t_{f_i} - t_{s_i})) / n$$

- 平均带权作业周转时间

$$W = (\sum_{i=1}^{n} w_i) / n = (\sum_{i=1}^{n} t_i / t_k) / n$$

3.2 调度算法

短程调度的决策模式

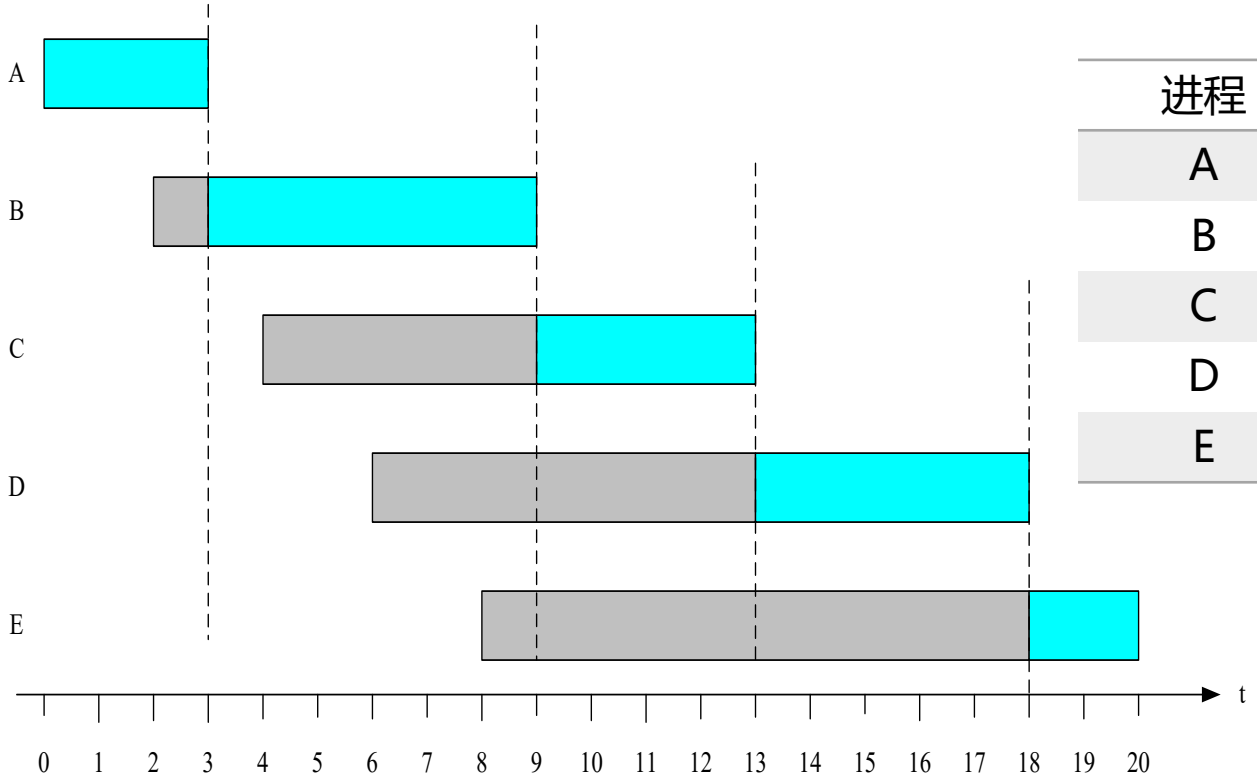
- 抢占式
  - 系统可以根据所规定的原则剥夺正在运行的进程/线程的处理器资源，将其移入就绪队列，选择其他进程/线程执行；
  - 剥夺原则
    - 高优先级进程/线程剥夺低优先级进程/线程
    - 当前运行进程/线程的时间片用完
- 非抢占式
  - 进程/线程开始运行后不再让出处理器，除非进程/线程运行结束，或因发生某个事件（如等待I/O操作完成）而不能继续执行。

3.2 调度算法—单处理器调度算法种类

- 先来先服务(First Come First Served: FCFS)
- 最短作业优先(Shortest Job First: SJF)
- 最短剩余时间优先(Shortest Remaining Time First: SRTF)
- 响应比最高者优先(Highest Response Ratio First: HRRF)
- 优先级调度
- 轮转调度(Round Robin: RR)
- 多级反馈队列调度(Multi-Level Feedback Queue: MLFQ)

先来先服务(First Come First Served: FCFS)

每个进程就绪时，加入就绪队列。当前进程停止执行时，选择在就绪队列中时间最长的进程执行。



进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

平均周转时间：(3+7+9+12+12)/5=8.6

平均带权周转时间：2.56

调度决策时刻：

在每个作业运行结束时决策

先来先服务(First Come First Served: FCFS)

- 优点：
  - 非抢占式调度算法，易于实现
  - 适用于作业调度和进程调度
- 缺点：
  - 效率不高
  - 不利于短作业而优待长作业
  - 不利于I/O繁忙作业而有利于CPU繁忙作业

先来先服务(First Come First Served: FCFS)

若进程到达顺序为1，2，3，则平均周转时间为(28+37+40)/3=35ms

若进程到达顺序为3，2，1，则平均周转时间为(3+12+40)/3≈18ms

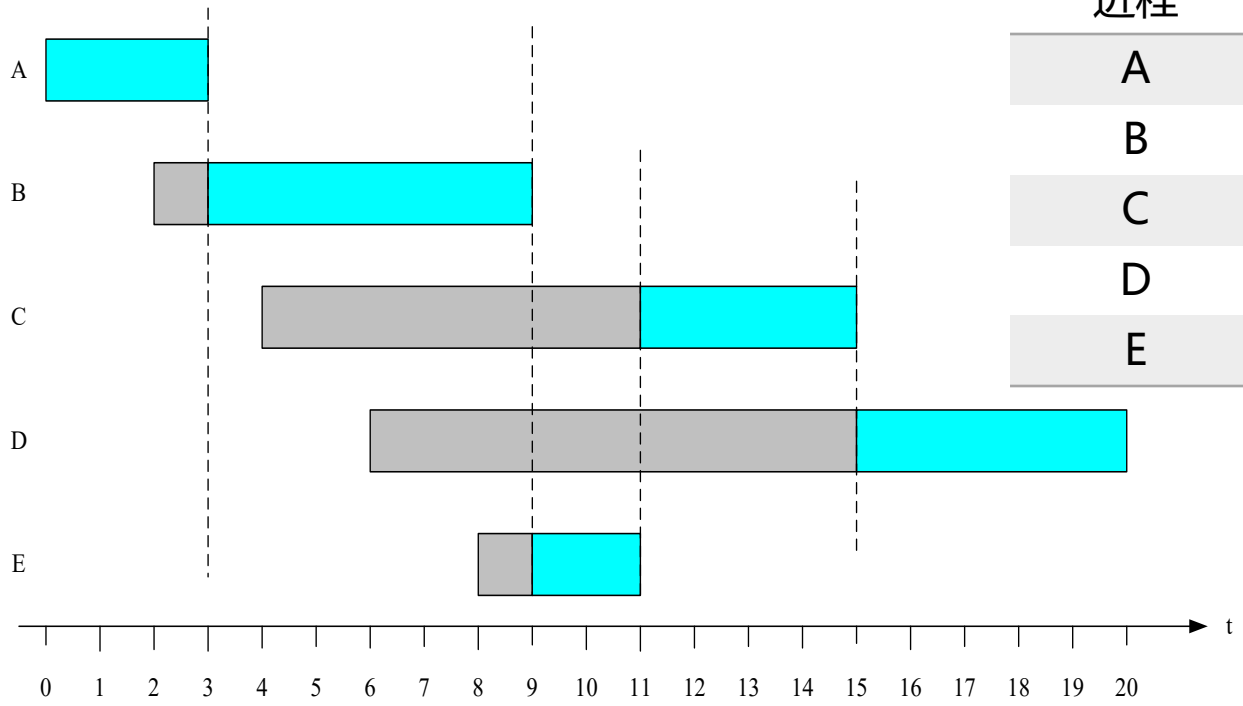
进程	所需CPU时间
进程1	28
进程2	9
进程3	3

最短作业优先(Shortest Job First: SJF)

- 机制
  - 当前进程停止执行时，选择预计所需的CPU运行时间最小的进程执行。
  - 非抢占式
- 优点：
  - 有利于短作业
  - 易于实现
- 缺点
  - 无法准确获知进程所需的CPU运行时间
  - 忽视作业的等待时间，有可能造成长作业饿死
  - 缺乏抢占机制，对分时、实时处理依然不理想。

最短作业优先(Shortest Job First: SJF)

当前进程停止执行时，选择预计所需的CPU运行时间最小的进程执行。 **非抢占式。**



进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

平均周转时间：(3+7+11+14+3)/5=7.6

平均带权周转时间：1.84

调度决策时刻：

在每个作业运行结束时决策

最短作业优先(Shortest Job First: SJF)

- 估算进程的下一个CPU周期长度（CPU突发期优先）
  - 指数衰减算法

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$$
$$= \alpha t_n + (1-\alpha)\alpha t_{n-1} + \cdots + (1-\alpha)^j \alpha t_{n-j} + \cdots + (1-\alpha)^n \tau_1$$

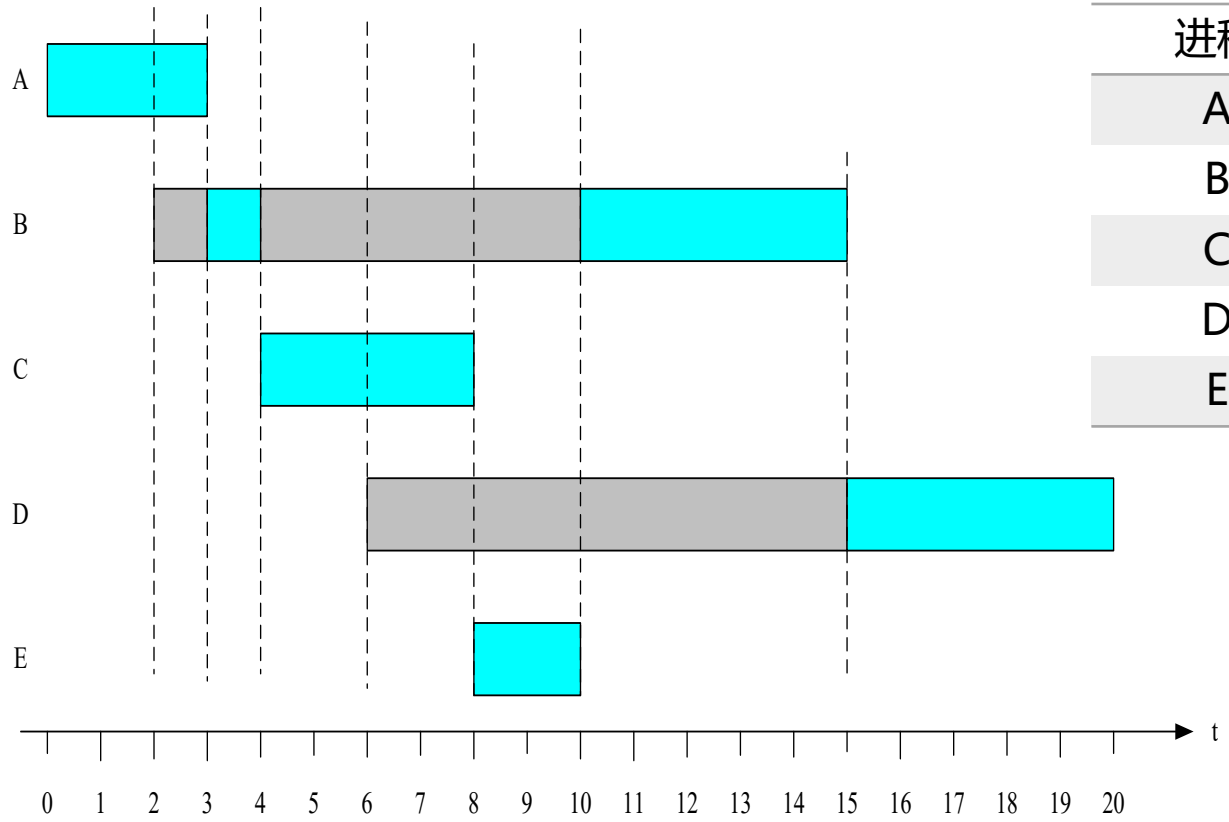
t_n 最近CPU突发周期

τ_n 是估算第n个CPU突发周期，它保存历史信息

最短剩余时间优先(Shortest Remaining Time First: SRTF)

- 机制：
  - 调度器总是选择具有**最短期望剩余运行时间**的进程运行；
  - 当一个新进程加入就绪队列时，它可能具有比当前运行进程更短的剩余运行时间，此时，调度器将让该**就绪进程抢占**当前运行的进程。
  - 实际上可以看作是支持处理器抢占的SJF
- 优点
  - 有利于短作业
  - 实现额外代价低
- 缺点
  - 必须估计进程的处理时间
  - 有可能会造成长作业的饿死

最短剩余时间优先(Shortest Remaining Time First: SRTF)



进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

平均周转时间：

(3+13+4+14+2)/5=7.2

平均带权周转时间：1.59

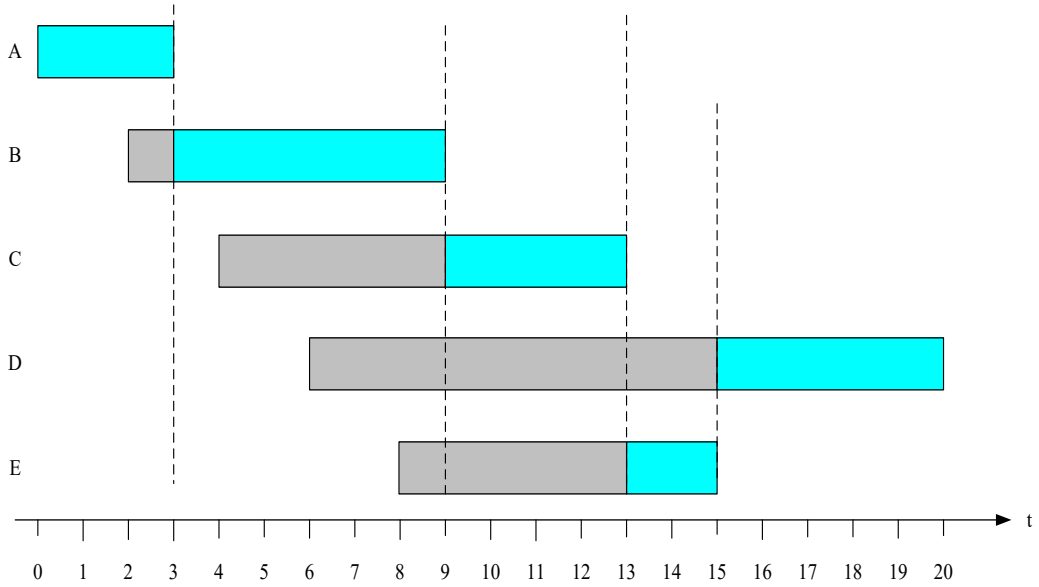
决策时刻：

（1）新作业到达时

（2）作业运行结束时

最高响应比优先(Highest Response Ratio First: HRRF)

- $R=(w+s)/s$；R: 响应比， w：**等待处理器的时间**，s：**服务时间**。
- 当前运行进程结束或阻塞时，选择响应比R值最大的进程执行。
- **非抢占式**



进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

决策时刻：**作业运行结束时**

平均周转时间：

(3+7+9+14+7)/5=8

平均带权周转时间：2.14

最高响应比优先(Highest Response Ratio First: HRRF)

- 优点
  - 考虑了进程的老化，有利于短进程(由于s小，所以R值将比较大)，也不会造成长进程的饿死(等待时间加长会使得其R值增加)
- 缺点
  - 需要估算进程的服务时间

优先级调度算法

- 优先级调度算法根据确定的优先级来选取进程/线程总是选取就绪队列中优先级最高者投入运行。
- 系统可以采用非剥夺式和剥夺式
- 规定优先级的方法

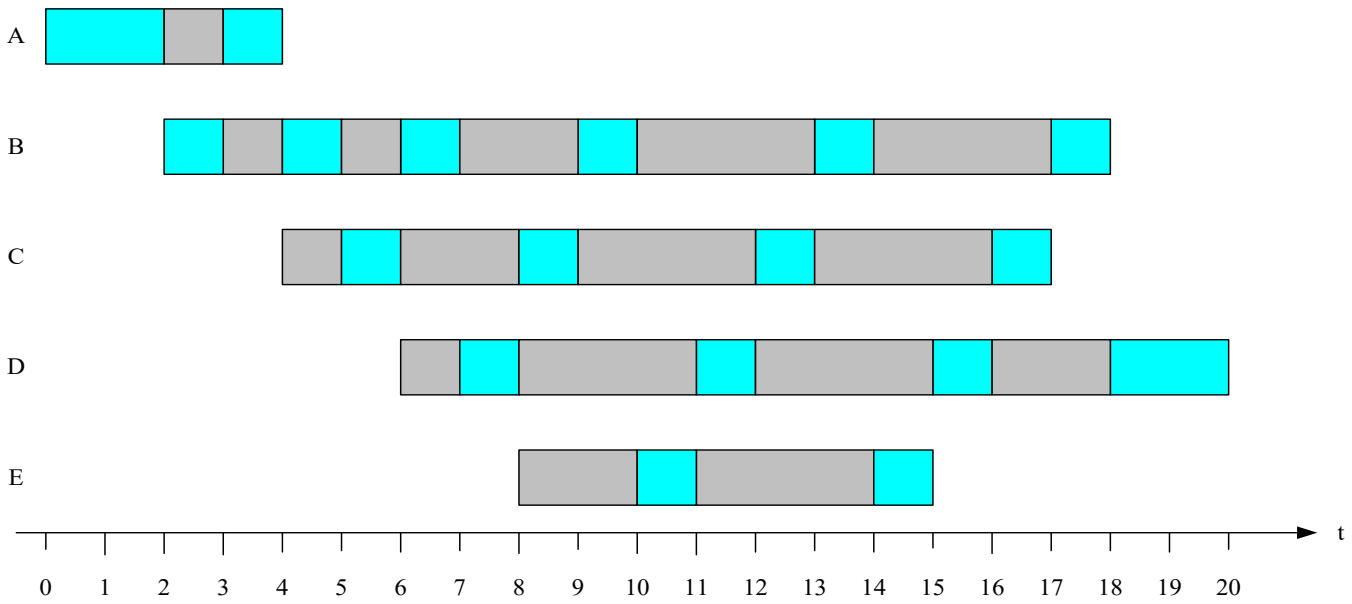
  用户提出优先级

  系统提出优先级
- 优先级确定分为静态和动态
- 静态：优先级在生命周期内不再改变->饥饿
- 动态：随着占有CPU时间增加，逐渐降低优先级；随着就绪队列中等待CPU的时间增加，逐渐提高优先级。

轮转调度(Round Robin: RR)

- 机制
  - 将时间划分成定长的时间片，当时间片完成后，产生时钟中断。
  - 当时钟中断发生后，当前运行进程被放到就绪队列，按FCFS的原则选取下一个就绪进程执行。
  - 抢占式
- 优点
  - 克服了FCFS中短作业可能会等待很长时间的问题
  - 有利于多用户、交互型进程
- 缺点
  - 增加了切换的额外开销
  - 对I/O密集型和CPU密集型作业一视同仁，优待了CPU密集型作业
- 时间片长度的选择对算法的性能影响很大

轮转调度(Round Robin: RR)



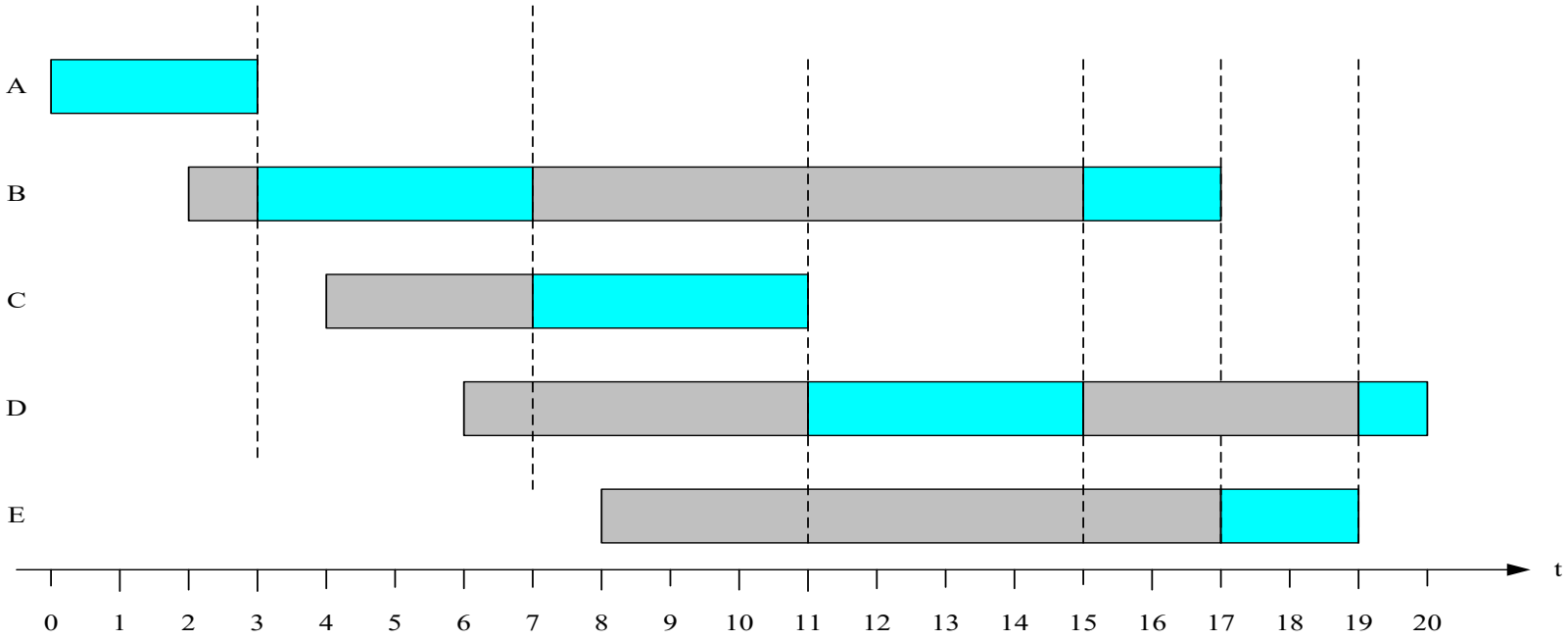
RR, 时间片长度为1

进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

平均周转时间

(4+16+13+14+7)/5=10.8

轮转调度(Round Robin: RR)



RR, 时间片长度为4

平均周转时间(3+15+7+14+11)/5=10

进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

多级反馈队列调度(Multi-Level Feedback Queue: MLFQ)

- 机制
  - SJF，SRTF，HRRF都需要预先知道进程的运行时间，这在实际上是很困难的。
  - MLFQ通过进程已经被服务的时间来预测进程的总服务时间。
  - MLFQ通过惩罚已经在处理器上运行时间很长的进程来达到优待短作业的效果。
  - 系统维持一个动态优先级队列RQ0, RQ1, RQ2,....。
  - 当进程首次进入系统，置于队列RQ0，当其运行时间片到并返回就绪队列时，被置于RQ1。随后每次由于时间片到被抢占，它将被置于下一级就绪队列。
  - 处理器每次先从第一个队列中选取执行执行者，每个队列内部采用FCFS的机制调度。只有未选到时才从较低一级的就绪队列中选取，仅当前面所有队列为空时才会运行最后一个就绪队列中的进程/线程。
  - 当进入最低优先级队列后，进程不能再进入更低优先级的队列，而是采用Round Robin的方式呆在该队列中，直至完成服务。

多级反馈队列调度(Multi-Level Feedback Queue: MLFQ)

- 优点
  - 无需预先估算进程的运行时间
  - 新进入系统的短进程将得到优待
- 缺点
  - 长进程可能会饿死
- MLFQ的变种
  - 进程每次允许被执行的时间片为定长
  - 处于队列i的进程被执行的时间片长度为2^i个时间单元
  - 若进程在当前队列等待时间过长后，可以提升到一个更高优先级的队列，从而避免长进程饿死

多级反馈队列调度(Multi-Level Feedback Queue: MLFQ)

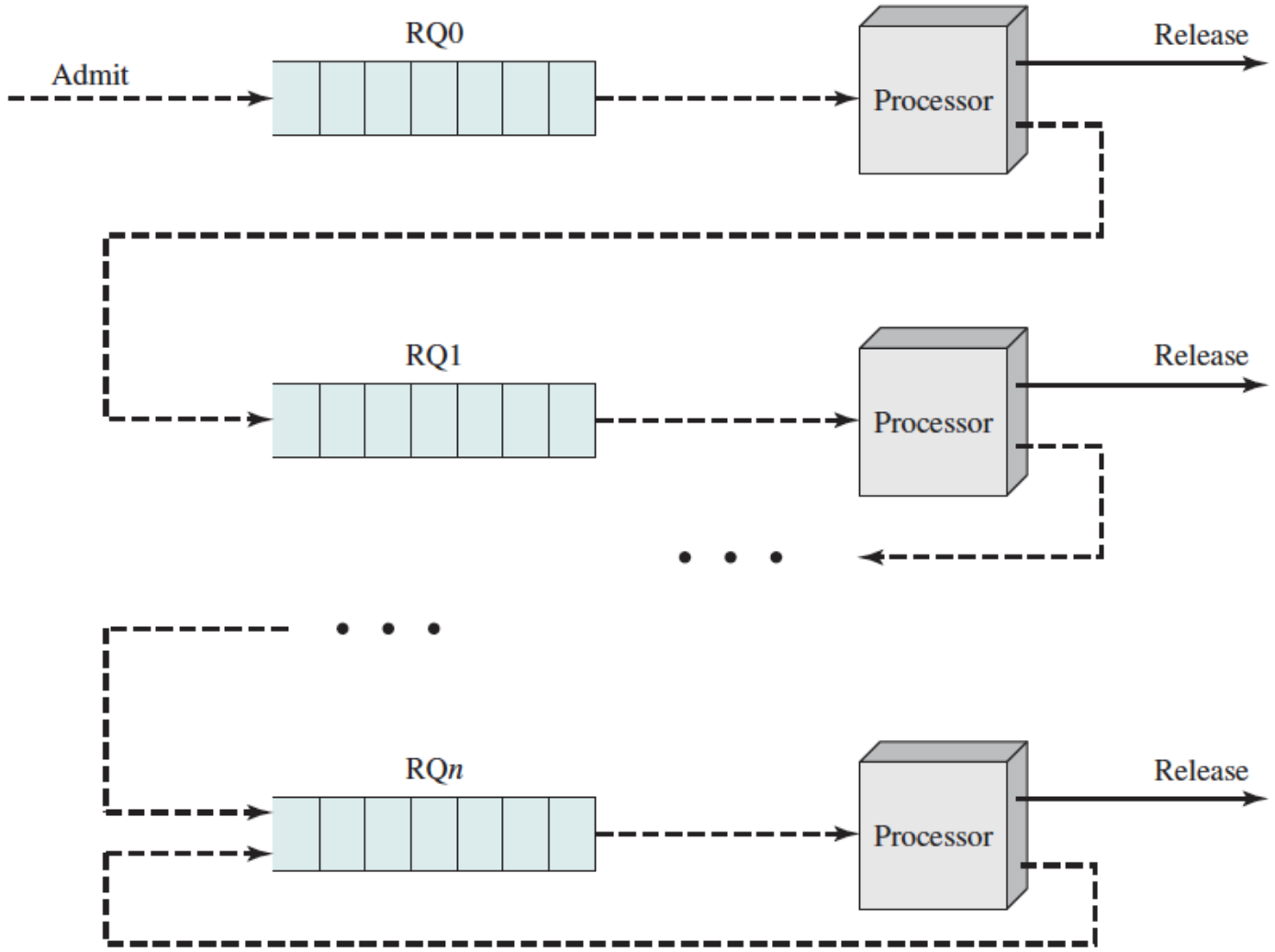
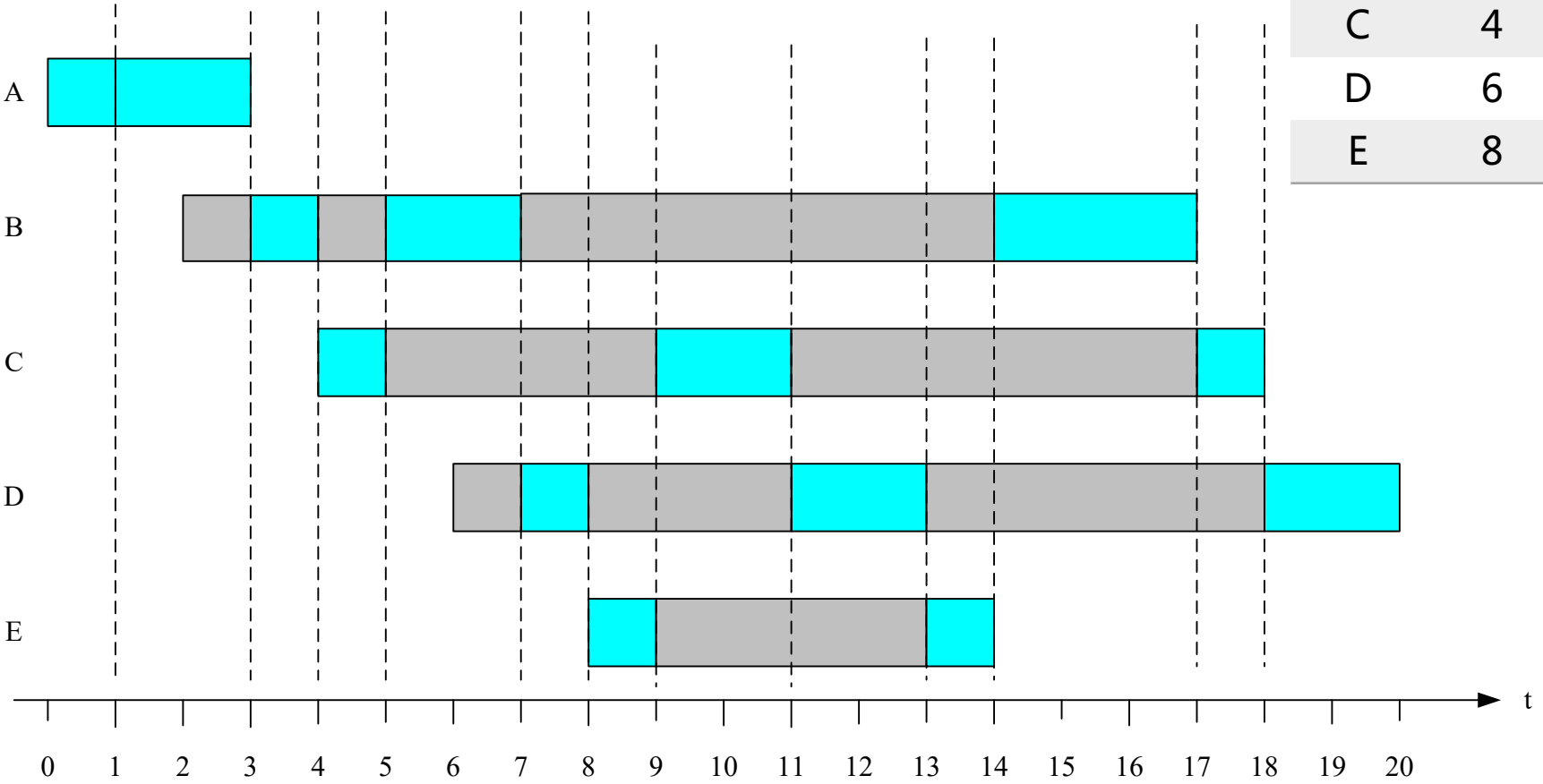


Figure 9.10 Feedback Scheduling

多级反馈队列调度(Multi-Level Feedback Queue: MLFQ)

进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



MLFQ,队列i的时间片为2^i时间单元

彩票调度算法

- 机制
  - 为进程发放针对系统各种资源（如CPU）的彩票，当调度程序需要作出决策时，随机选择一张彩票，持有该彩票的进程将获得系统资源。
  - 对CPU调度，系统可能每秒钟抽50次彩票，每次中奖者可以获得20ms的运行时间。
- 优点
  - 所有进程都是平等的，有相同的运行机会
  - 如果某些进程需要更多机会，可以被赋予更多的额外彩票。

Linux中的CFS(Completely Fair Scheduling)调度算法

- 两个层面的优先级
  - nice值(-20~19),默认取值0, 越大表明优先级越小，获得的CPU份额越少。
    - ps –el
  - 实时优先级，默认取值范围[0,99]，值越大，优先级越高。
    - ps –eo 显示-表示进程不是实时进程

  调度相关源代码可以在kernel/sched/目录下找到

CFS调度算法

- 处理器的时间片长度是当前系统负载的函数
  - 负载越高，时间片长度越短
- 每个进程的时间片长度进一步根据进程的nice值进行加权调节。
- 抢占式调度
  - 每当一个进程就绪，是否抢占当前正在运行的进程，取决于两者已经运行的被分配时间片的比例大小。

实时调度

- 硬实时
  - 必须满足时间限制
- 软实时
  - 偶尔超过时间限制是可以容忍的
- 响应事件分类
  - 周期性事件
  - 非周期性事件

实时调度算法

- 单比例调度算法
  - 基于周期长度的抢占式调度策略
  - 周期越短，优先级越高
    - 如，为每个进程/线程分配与事件发生频率成正比的优先数
- 限期调度算法
  - 将进程/线程按照发生事件的截止处理期限排序
  - 周期性事件的截止期限为事件下一次发生的时间
  - 选择截止期限最近的进程/进程运行
  - 当新进程/线程就绪时，依据截止期限决定是否抢占当前运行进程/线程
- 最少裕度法
  - 计算各个进程/线程的富裕时间（裕度），然后选择裕度最少者执行

    裕度=截止时间-(就绪时间+计算时长)

3.3 多处理器调度

- 多处理器系统的类型
  - 非对称多处理器(asymmetric multiprocessing)
    - Master server: 负责所有的调度决策、I/O处理、以及其它系统行为
    - Slave client: 仅执行用户代码
  - 对称多处理器(symmetric multiprocessing)
    - 每个处理器都具有相同的角色，自我调度
    - 所有处理器共享公共的就绪队列 or 每个处理器有自己的私有就绪队列

3.3 多处理器调度—决策因素

- 处理器亲和性(processor affinity)
  - 由于高速缓存的存在，如果一个进程在执行过程中被调度到其它处理器继续执行，则之前被缓存在高速缓存中与该进程相关的数据都将失效
  - 因此，应尽可能让进程在一个处理器上完成其执行，而会在生命周期内在不同的处理器上迁移。
  - 实现上存在两种模式
    - Soft affinity
    - Hard affinity
- 负载均衡(load balancing)
  - 试图将工作负载均匀地分布到不同的处理器上
  - 仅当每个处理器都有一个私有的就绪队列时才成为问题（共享公共就绪队列不存在负载均衡问题）
  - 实现方式：
    - Push migration： 存在一个进程定期检查每个处理器的负载
    - Pull migration：由空闲处理器主动从繁忙的处理器取工作负载
  - 负载均衡与处理器亲和性存在矛盾

3.3 多处理器调度—调度算法

- 负载共享调度算法
  - 实现方法
    - 系统维护全局性进程就绪队列
    - 当处理器空闲时，就选择进程的一个线程去运行
  - 优点
    - 负载均衡
    - 无需集中调度
  - 缺点
    - 就绪队列必须被互斥访问
    - 违背了处理器亲和性
- 群调度算法
  - 一群相关线程被同时调度到一组处理器上运行
  - 紧密相关线程的并行执行能够减少同步阻塞，从而减少进程切换，降低调度代价，提高系统性能
- 专用处理器调度算法
  - 将同属一个进程的一组线程同时分派到一组处理机上运行，每个线程均获得一个处理机，专用于处理这个线程，直到进程运行结束。
  - 是群调度的一种极端形式
  - 当线程因等待事件而阻塞时，并不让出处理器，这就是所谓的专用性