上节课重点：

1. 进程通信: 低级通信和高级通信。

2. 线程：一个基本的CPU执行单元，也是程序执行流的最小单位。处理机的分配单元。

3. 处理机调度的层次：高级调度（作业调度）、中级调度（内存调度）、低级调度（进程调度）.

4. 调度队列模型和调度准则：CPU利用率、系统吞吐量、周转时间、等待时间和响应时间等。

3.2 作业与作业调度

- OS中调度的实质是一种资源分配，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。
- 有的调度算法适用于作业调度，有的算法适用于进程调度，有的两者都适应。

一、先来先服务和短作业（进程）优先调度算法
二、高优先权优先调度算法

3.2.1先来先服务和短作业优先调度算法

- FCFS算法
  - 算法描述
    - 按照作业提交或进程变为就绪状态的先后次序，分派CPU。当前作业或进程占用CPU，直到执行完或阻塞，才出让CPU（非抢占方式）。
    - 在作业或进程唤醒后（如I/O完成），并不立即恢复执行，而是进入就绪队列排队。
    - 最简单的算法。
  - FCFS的特点
    - 比较有利于长作业，而不利于短作业。
    - 有利于CPU繁忙的作业，而不利于I/O繁忙的作业。

3.2.1先来先服务和短作业优先调度算法

下表列出了A、B、C、D四个作业分别到达系统的时间、要求服务的时间、开始执行的时间及各自的完成时间，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				
B	1	100				
C	2	1				
D	3	100				

从表上可以看出，其中短作业C的带权周转时间竟高达100，这是不能容忍的；而长作业D的带权周转时间仅为1.99。FCFS调度算法比较有利于长作业，而不利于短作业；有利于CPU繁忙型的作业，而不利于I/O繁忙型的作业（进程）

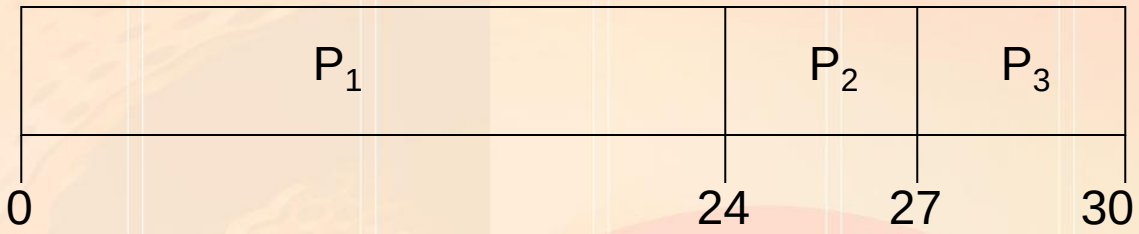
CPU繁忙型作业：如通常的科学计算。
I/O繁忙型作业：指CPU进行处理时，需频繁的请求I/O。

3.2.1先来先服务和短作业优先调度算法

例：

进程	到达时间	执行时间
P_1	0.0	24
P_2	1.0	3
P_3	2.0	3

- Gantt 图:



- 平均等待时间： (0 + 23 + 25)/3 = 16
- 平均周转时间：(24+26+28)/3=26
- 平均带权周转时间： (24/24+26/3+28/3)/3=6.33

3.2.1先来先服务和短作业优先调度算法

FCFS

算法思想	主要从“公平”的角度考虑（类似于我们生活中排队买东西的例子）
算法规则	按照作业/进程到达的先后顺序进行服务
用于作业/进程调度	用于作业调度时，考虑的是哪个作业先到达后备队列；用于进程调度时，考虑的是哪个进程先到达就绪队列
是否可抢占？	非抢占式的算法
优缺点	优点：公平、算法实现简单 缺点：排在长作业（进程）后面的短作业需要等待很长时间，带权周转时间很大，对短作业来说用户体验不好。即，FCFS算法对长作业有利，对短作业不利（Eg：排队买奶茶...）
是否会导致饥饿（某进程/作业长期得不到服务）	不会

3.2.1先来先服务和短作业优先调度算法

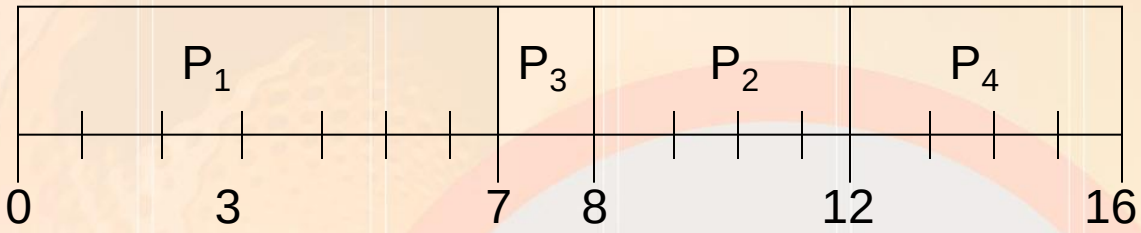
- 短作业（进程）优先调度算法(Shortest Job/Process First，SJF/SPF)
  - 算法描述
    - 对预计执行时间短的作业（进程）优先分派处理机。通常后来的短作业不抢先正在执行的作业。
    - 是对FCFS算法的改进，其目标是减少平均周转时间。
  - SJF的特点
    - 优点：
      - *比FCFS改善平均周转时间和平均带权周转时间，缩短作业的等待时间；*
      - *提高系统的吞吐量；*
    - 缺点：
      - *对长作业非常不利，可能长时间得不到执行；*
      - *未能依据作业的紧迫程度来划分执行的优先级；*
      - *难以准确估计作业（进程）的执行时间，从而影响调度性能。*

作业情况 / 调度算法	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						

采用SJF算法后，不论是平均周转时间还是平均带权周转时间，都较FCFS调度算法有明显的改善，尤其是对短作业D。而平均带权周转时间从2.8降到了2.1。这说明SJF调度算法能有效的降低作业的平均等待时间，提高系统吞吐量。

进程	到达时间	执行时间
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- 非抢先式SJF



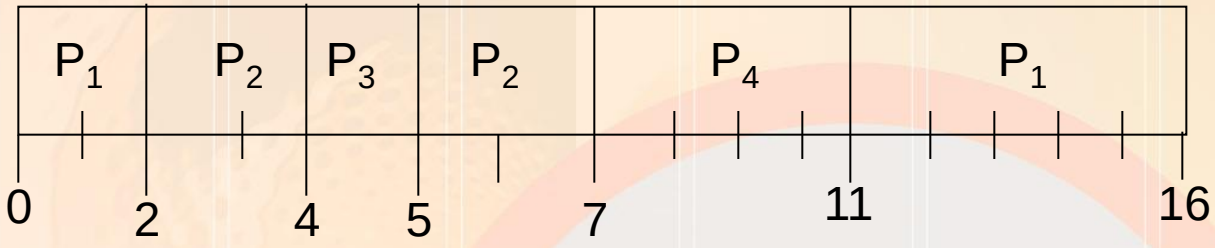
- 平均等待时间 = (0 + 6 + 3 + 7)/4 = 4
- 平均周转时间 = (7 + 4 + 10 ＋ 11)/4= 8
- 平均带权周转时间=

3.2.1先来先服务和短作业优先调度算法

- SJF的变型
  - “最短剩余时间优先”SRT(Shortest Remaining Time)（允许比当前进程剩余时间更短的进程来抢占）
  - “最高响应比优先”HRRN(Highest Response Ratio Next)（响应比R = (等待时间 + 要求执行时间) / 要求执行时间，是FCFS和SJF的折衷）

进程	到达时间	执行时间
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- 最短剩余时间优先（抢先式SJF）



- 平均等待时间 = (9 + 1 + 0 +2)/4 = 3
- 平均周转时间＝（16＋5＋1＋6）/4=7

短作业优先(SJF)

注意几个小细节：

1. 如果题目中未特别说明，所提到的“短作业/进程优先算法”默认是非抢占式的

2. 很多书上都会说“SJF 调度算法的平均等待时间、平均周转时间最少”
严格来说，这个表述是错误的，不严谨的。之前的例子表明，最短剩余时间优先算法得到的平均等待时间、平均周转时间还要更少
应该加上一个条件“在所有进程同时可运行时，采用SJF调度算法的平均等待时间、平均周转时间最少”；
或者说“在所有进程都几乎同时到达时，采用SJF调度算法的平均等待时间、平均周转时间最少”；

如果不加上述前提条件，则应该说“抢占式的短作业/进程优先调度算法（最短剩余时间优先, SRNT算法）的平均等待时间、平均周转时间最少”

3. 虽然严格来说，SJF的平均等待时间、平均周转时间并不一定最少，但相比于其他算法（如 FCFS），SJF依然可以获得较少的平均等待时间、平均周转时间

4. 如果选择题中遇到“SJF 算法的平均等待时间、平均周转时间最少”的选项，那最好判断其他选项是不是有很明显的错误，如果没有更合适的选项，那也应该选择该选项

短作业优先(SJF)

SJF

算法思想	追求最少的平均等待时间，最少的平均周转时间、最少的平均平均带权周转时间
算法规则	最短的作业/进程优先得到服务（所谓“最短”，是指要求服务时间最短）
用于作业/进程调度	即可用于作业调度，也可用于进程调度。用于进程调度时称为“短进程优先（SPF, Shortest Process First）算法”
是否可抢占？	SJF和SPF是非抢占式的算法。但是也有抢占式的版本——最短剩余时间优先算法（SRTN, Shortest Remaining Time Next）
优缺点	优点：“最短的”平均等待时间、平均周转时间 缺点：不公平。对短作业有利，对长作业不利。可能产生饥饿现象。另外，作业/进程的运行时间是由用户提供的，并不一定真实，不一定能做到真正的短作业优先
是否会导致饥饿	会。如果源源不断地有短作业/进程到来，可能使长作业/进程长时间得不到服务，产生“饥饿”现象。如果一直得不到服务，则称为“饿死”

对FCFS和SJF两种算法的思考...

FCFS 算法是在每次调度的时候选择一个等待时间最长的作业（进程）为其服务。但是没有考虑到作业的运行时间，因此导致了对短作业不友好的问题

SJF 算法是选择一个执行时间最短的作业为其服务。但是又完全不考虑各个作业的等待时间，因此导致了对长作业不友好的问题，甚至还会造成饥饿问题

能不能设计一个算法，即考虑到各个作业的等待时间，也能兼顾运行时间呢？

高响应比优先算法

高响应比优先调度算法 (Highest Response Ratio Next，HRRN)

算法思想：要综合考虑作业/进程的等待时间和要求服务的时间

高响应比优先算法：非抢占式的调度算法，只有当前运行的进程主动放弃CPU时(正常/异常完成，或主动阻塞)，才需要进行调度。在每次调度时，计算所有就绪进程的响应比，选择响应比最高的进程上处理机。

- 响应比R
  =（等待时间 + 要求执行时间）/ 要求执行时间
  ＝1＋等待时间/要求执行时间

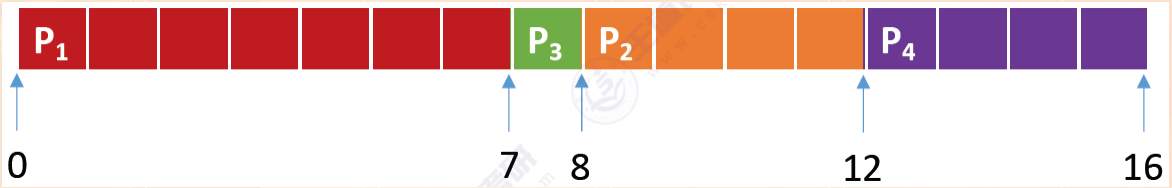
- 是FCFS和SJF的折衷：
  - 作业等待时间相同，服务时间越短，优先权越高--SJF；
  - 要求服务时间相同，等待时间越长，优先权越高--FCFS；长作业随着等待时间的增加，优先权增加。
- 缺点：
  - 响应比的计算增加系统开销

高响应比优先调度算法 (Highest Response Ratio Next， HRRN)

例题：各进程到达就绪队列时间、需要的运行时间如下表所示。使用高响应比优先调度算法，计算各进程的等待时间、平均等待时间、周转时间、平均周转时间、带权周转时间、平均带权周转时间。

进程	到达时间	运行时间
P1	0	7
P2	2	4
P3	4	1
P4	5	4

高响应比优先算法：非抢占式的调度算法，只有当前运行的进程主动放弃CPU时(正常/异常完成，或主动阻塞)，才需要进行调度，调度时计算所有就绪进程的响应比，选择响应比最高的进程上处理机。



$$相应比 = \frac{等待时间 + 要求服务时间}{要求服务时间}$$

P2和P4要求服务时间一样，但P2等待时间长，所以必然是P2响应比更大

0时刻：只有P1到达就绪队列，P1上处理机。

7时刻(P1主动放弃CPU)：就绪队列中有P2(响应比=(5+4)/4=2.25)，P3(响应比=(3+1)/1=4)
P4(响应比=(2+4)/4=1.5)

8时刻(P3完成)：P2(2.5)、P4(1.75)

12时刻(P2完成)：就绪队列中只剩下P4

高响应比优先调度算法 (Highest Response Ratio Next，HRRN)

- 例：如表所示四个作业进入系统，分别计算HRRN算法下的平均周转时间和带权周转时间。

作业	提交时间（时）	运行时间（分）
1	8:00	120
2	8:50	50
3	9:00	10
4	9:50	20

Job1完成后的响应比：

Job2: 70/50=1.4

Job3: 60/10=6

Job4: 10/20=0.5

所以调度Job3执行。

Job3完成后的响应比：

Job2: 80/50=1.8

Job4: 20/20=1

所以调度Job2执行

作业	开始时间	完成时间	周转时间
1	8:00	10:00	120
2	10:10	11:00	130
3	10:00	10:10	70
4	11:00	11:20	90

平均周转时间＝**102.5**

带权周转时间**W= T(周转)/ (CPU执行)**

平均带权周转时间＝ **(120/120 + 130/50 + 70/10 + 90/20)/4=3.775**

高响应比优先调度算法 (Highest Response Ratio Next，HRRN，HRN)

HRRN

算法思想	要综合考虑作业/进程的等待时间和要求服务的时间
算法规则	在每次调度时先计算各个作业/进程的响应比，选择响应比最高的作业/进程为其服务 响应比 = $\frac{\text{等待时间+要求服务时间}}{\text{要求服务时间}}$ 响应比 ≥ 1
用于作业/进程调度	即可用于作业调度，也可用于进程调度
是否可抢占？	非抢占式的算法。因此只有当前运行的作业/进程主动放弃处理机时，才需要调度，才需要计算响应比
优缺点	综合考虑了等待时间和运行时间（要求服务时间） 等待时间相同时，要求服务时间短的优先（SJF 的优点） 要求服务时间相同时，等待时间长的优先（FCFS 的优点） 对于长作业来说，随着等待时间越来越久，其响应比也会越来越大，从而避免了长作业饥饿的问题
是否会导致饥饿	不会

- 例：满足短作业优先且不会发生饥饿现象的调度算法是：

- A FCFS　　B 高响应比优先
- C 时间片轮转
- D 非抢先式SJF

知识回顾与重要考点

算法	思想&规则	可抢占？	优点	缺点	考虑到等待时间&运行时间？	会导致饥饿？
FCFS	自己回忆	非抢占式	公平；实现简单	对短作业不利	等待时间√ 运行时间×	不会
SJF/SPF	自己回忆	默认为非抢占式，也有SJF的抢占式版本最短剩余时间优先算法（SRTN）	“最短的”平均等待/周转时间；	对长作业不利，可能导致饥饿；难以做到真正的短作业优先	等待时间× 运行时间√	会
HRRN	自己回忆	非抢占式	上述两种算法的权衡折中，综合考虑的等待时间和运行时间		等待时间√ 运行时间√	不会

注：这几种算法主要关心对用户的公平性、平均周转时间、平均等待时间等评价系统整体性能的指标，但是不关心“响应时间”，也并不区分任务的紧急程度，因此对于用户来说，交互性很糟糕。因此这三种算法一般适合用于早期的批处理系统，当然，FCFS算法也常结合其他的算法使用，在现在也扮演着很重要的角色。而适合用于交互式系统的调度算法将在下个小节介绍...

提示：一定要动手做课后习题！这些算法特性容易考小题，算法的使用常结合调度算法的评价指标在大题中考察。

- 时间片轮转法（Round Robin, RR）
  - 本算法主要用于微观调度（进程调度）
  - 设计目标是提高资源利用率
  - 基本思路是通过时间片轮转，提高进程并发性和响应时间特性，从而提高资源利用率
  - 是否可抢占？若进程未能在时间片内运行完，将被强行剥夺处理机使用权，因此时间片轮转算法属于抢占式的算法，由时钟装置发出时钟中断来通知CPU时间片已到

时间片轮转算法

- 算法描述
  - 将系统中所有的就绪进程按照FCFS原则，排成一个队列。
  - 每次调度时将CPU分派给队首进程，让其执行一个时间片。在一个时间片结束时，发生时钟中断。调度程序据此暂停当前进程的执行，将其送到就绪队列的末尾，并通过上下文切换执行当前的队首进程。
  - 时间片的长度从几个ms到几百ms。
  - 进程可以未使用完一个时间片，就出让CPU（如阻塞）。
- 时间片长度的确定
  - 时间片长度变化的影响
    - 过长—>退化为FCFS算法
    - 过短—>用户的一次请求需要多个时间片才能处理完，上下文切换次数增加，响应时间长。
  - 就绪进程的数目：数目越多，时间片越小
  - 系统的处理能力：应当使用户输入通常在一个时间片内能处理完，否则使响应时间，平均周转时间和平均带权周转时间延长。

时间片轮转算法

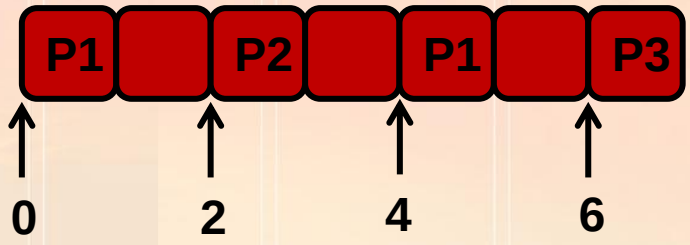
- 例：降低进程优先级的合理时机是
  - A 进程的时间片用完
  - B 进程刚完成IO，进入就绪队列
  - C 进程长期处于就绪队列中
  - D 进程从就绪态转为运行态

时间片轮转算法

例题：各进程到达就绪队列的时间、需要的运行时间如下表所示。使用时间片轮转调度算法，分析时间片大小分别是2、5时的进程运行情况。

进程	到达时间	运行时间
P_1	0	5
P_2	2	4
P_3	4	1
P_4	5	6

时间片轮转调度算法：轮流让就绪队列中的进程依次执行一个时间片(每次选择的都是排在就绪队列队头的进程)。



时间片大小为2(注：以下括号内表示当前时刻就绪队列中的进程、进程的剩余时间)

0时刻(P1(5)):0时刻只有P1到达就绪队列，让P1上处理机运行一个时间片

2时刻(P2(4)→P1(3)):2时刻P2到达就绪队列，P1运行完一个时间片，被剥夺处理机，重新放回队尾。此时P2排在队头，因此让P2上处理机。

4时刻(P1(3)→P3(1)→P2(2)): 4时刻，P3到达，先插到就绪队尾，紧接着，P(2)下处理机也插到队尾。

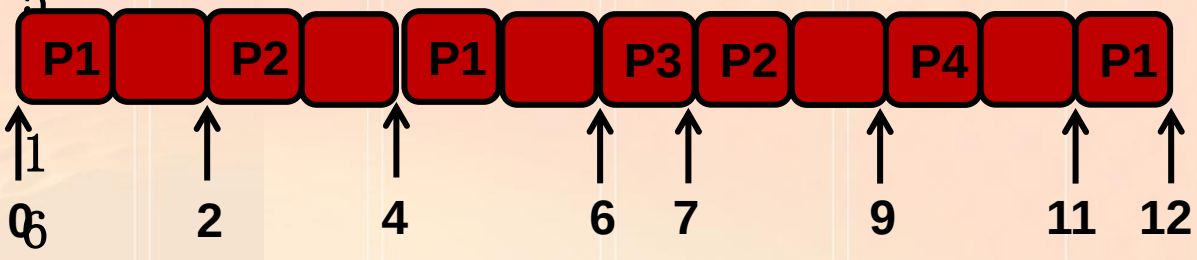
5时刻(P1(3)→P3(1)→P2(2)→P4(6)): 5时刻，P4到达插入到就绪队尾(注意：由于P1的时间片还没用完，因此暂时不调度。另外，此时P1处于运行态，并不在就绪队列中)

时间片轮转算法

例题：各进程到达就绪队列的时间、需要的运行时间如下表所示。使用时间片轮转调度算法，分析时间片大小分别是2、5时的进程运行情况。

进程	到达时间	运行时间
P_1	0	5
P_2	2	2
P_3	4	1
P_4	6	6

时间片轮转调度算法：轮流让就绪队列中的进程依次执行一个时间片(每次选择的都是排在就绪队列队头的进程)。



时间片大小为2(注：以下括号内表示当前时刻就绪队列中的进程、进程的剩余时间)

6时刻(P3(1)→P2(2)→P4(6)→P1(1)):6时刻，P1时间片用完，下处理机。重新放回就绪队尾，发生调度。

7时刻(P2(2)→P4(6)→P1(1)):虽然P3时间片没用完，但是由于P3只需运行1一个单位时间，运行完了会主动放弃处理机，因此也会发生调度。队头进程P2上处理机。

9时刻(P4(6)→P1(1)): 进程P2时间片用完，并刚好运行完，发生调度，P4上处理机。

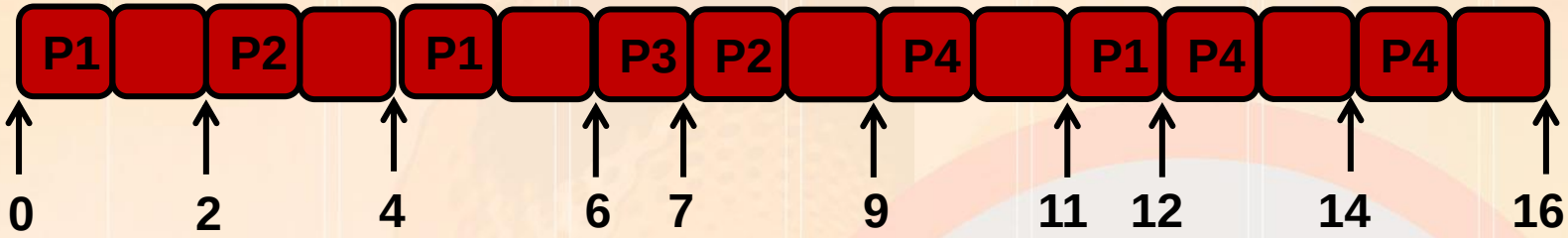
11时刻(P1(1)→P4(4)): P4时间片用完，重新回到就绪队列。P1上处理机。

时间片轮转算法

例题：各进程到达就绪队列的时间、需要的运行时间如下表所示。使用时间片轮转调度算法，分析时间片大小分别是2、5时的进程运行情况。

进程	到达时间	运行时间
P_1	0	5
P_2	2	4
P_3	4	1
P_4	5	6

时间片轮转调度算法：轮流让就绪队列中的进程依次执行一个时间片(每次选择的都是排在就绪队列队头的进程)。



时间片大小为2(注：以下括号内表示当前时刻就绪队列中的进程、进程的剩余时间)

12时刻(P4(4)):P1运行完，主动放弃处理机，此时就绪队列中只剩P4，P4上处理机。

14时刻(P4(2)):就绪队列为空，因此让P4接着运行一个时间片。

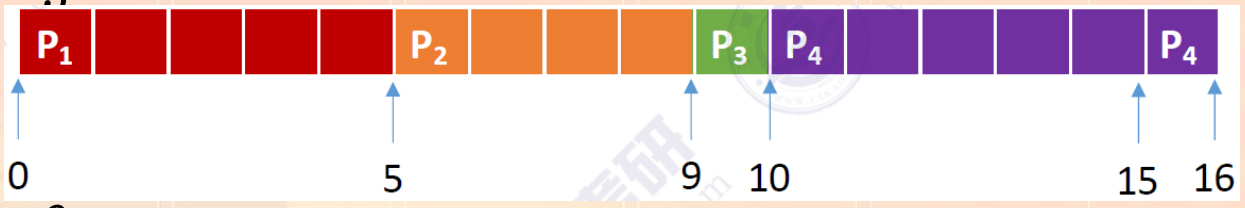
16时刻：所有进程运行结束。

时间片轮转算法

例题：各进程到达就绪队列的时间、需要的运行时间如下表所示。使用时间片轮转调度算法，分析时间片大小分别是2、5时的进程运行情况。

时间片轮转调度算法：轮流让就绪队列中的进程依次执行一个时间片(每次选择的都是排在就绪队列队头的进程)。

进程	到达时间	运行时间
P_1	0	5
P_2	2	2
P_3	4	1
P_4	5	6



时间片大小为5(注：以下括号内表示当前时刻就绪队列中的进程、进程的剩余时间)

0时刻(P1(5)):0时刻只有P1到达就绪队列，让P1上处理机运行一个时间片。

2时刻(P2(4)):P2到达，但P2时间片尚未结束，因此暂不调度。

4时刻(P2(4)→P3(1)): P3到达，但P1时间片尚未结束，因此暂不调度。

5时刻(P2(4)→P3(1)→P4(6)):P4到达，同时，P1运行结束。发生调度，P2上处理机。

9时刻(P3(1)→P4(6)): P2运行结束，虽然时间片没用完，但是会主动放弃处理机。发生调度。

10时刻(P4(6)): P3运行结束，虽然时间片没用完，但是会主动放弃处理机，发生调度。

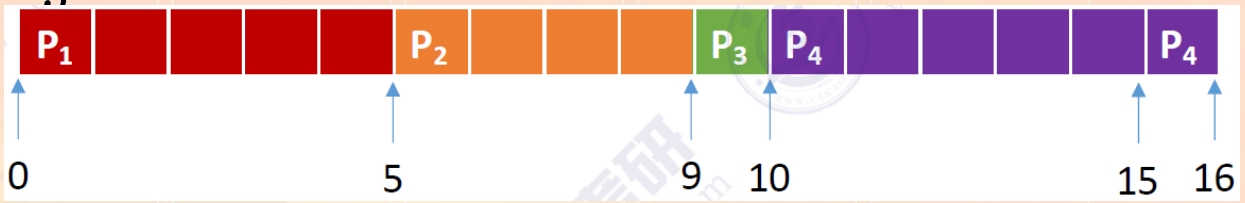
15时刻(): P4时间片用完，但就绪队列为空，因此会让P4继续执行一个时间片。

时间片轮转算法

例题：各进程到达就绪队列的时间、需要的运行时间如下表所示。使用时间片轮转调度算法，分析时间片大小分别是2、5时的进程运行情况。

进程	到达时间	运行时间
P_1	0	5
P_2	2	6
P_3	4	1
P_4	5	4

时间片轮转调度算法：轮流让就绪队列中的进程依次执行一个时间片(每次选择的都是排在就绪队列队头的进程)。



若按照先来先服务调度算法...

如果时间片太大，使得进程都可以在一个时间片内就完成，则时间片轮转调度算法退化为先来先服务调度算法，并且会增大进程响应时间。因此时间片不能太大。

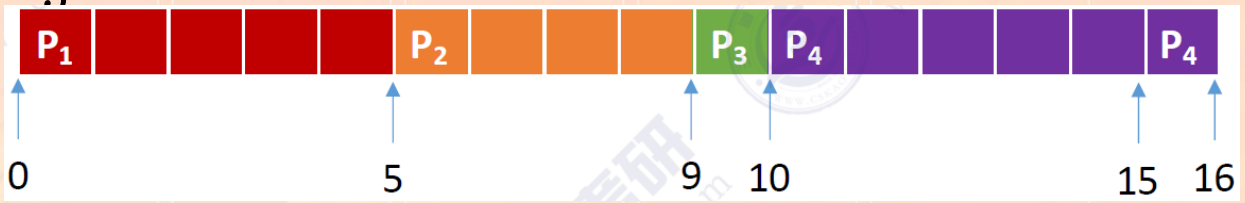
比如：系统中有10个进程在并发执行，如果时间片为1秒，则一个进程被响应可能需要等待9秒...也就是说，如果用户在自己进程的时间片外通过键盘发出调试命令，可能需要等待9秒才能被系统响应。

时间片轮转算法

例题：各进程到达就绪队列的时间、需要的运行时间如下表所示。使用时间片轮转调度算法，分析时间片大小分别是2、5时的进程运行情况。

进程	到达时间	运行时间
P_1	0	5
P_2	2	2
P_3	4	1
P_4	5	6

时间片轮转调度算法：轮流让就绪队列中的进程依次执行一个时间片(每次选择的都是排在就绪队列队头的进程)。



若按照先来先服务调度算法...

如果时间片太大，使得进程都可以在一个时间片内就完成，则时间片轮转调度算法退化为先来先服务调度算法，并且会增大进程响应时间。因此时间片不能太大。

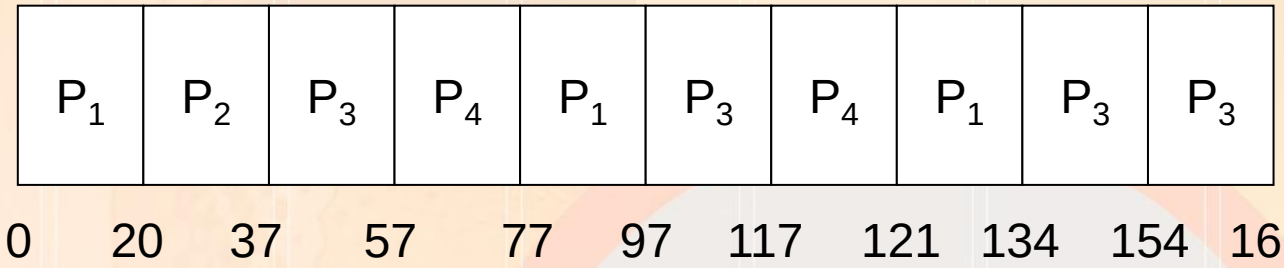
另一方面，进程调度、切换是有时间代价的（保存、恢复运行环境），因此如果时间片太小，会导致进程切换过于频繁，系统会花大量的时间来处理进程切换，从而导致实际用于进程执行的时间比例减少。可见时间片也不能太小。

一般来说，设计时间片时要让切换进程的开销占比不超过1%。

轮转调度算法（时间片＝20）

进程	执行时间
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



轮转调度算法（时间片＝20）

进程	到达时间	执行时间
P_1	0	3
P_2	1	6
P_3	4	4
P_4	6	2

时间片 ＝ 2

P1	P2	P3	P4	P1	P2	P3	P2
2	4	6	8	9	11	13	15

时间片轮转算法

时间片轮转（RR, Round-Robin）

优先级调度

算法思想	公平地、轮流地为各个进程服务，让每个进程在一定时间间隔内都可以得到响应
算法规则	按照各进程到达就绪队列的顺序，轮流让各个进程执行一个时间片（如 100ms）。若进程未在一个时间片内执行完，则剥夺处理机，将进程重新放到就绪队列队尾重新排队。
用于作业/进程调度	用于进程调度（只有作业放入内存建立了相应的进程后，才能被分配处理机时间片）
是否可抢占？	若进程未能在时间片内运行完，将被强行剥夺处理机使用权，因此时间片轮转调度算法属于抢占式的算法。由时钟装置发出时钟中断来通知CPU时间片已到
优缺点	优点：公平；响应快，适用于分时操作系统； 缺点：由于高频率的进程切换，因此有一定开销；不区分任务的紧急程度。
是否会导致饥饿	不会
补充	时间片太大或太小分别有什么影响？

优先权调度算法(Priority Scheduling)

- 本算法适用于作业调度和进程调度。
- 算法用于作业调度时，系统从后备队列中选择优先权最高的作业装入内存。
- 算法用于进程调度时，系统把处理机派发给就绪队列中优先权最高的进程。

优先权调度算法(Priority Scheduling)

- 优先权调度算法的类型分为抢占式和非抢占式
  - 非抢占式方式：除非自愿或时间片到，当前的进程不可以被优先级更高的进程抢用CPU。
  - 抢占方式：当前的进程在其时间片未用完时就可被优先级更高的进程抢用CPU（自己则进入就绪状态）。
  - 可抢占程度越高，对实时系统的满足越好。

优先权调度算法(Priority Scheduling)

- 完全不可抢占或用户态不可抢占：无论在用户态或核心态下执行代码，都不可被抢用CPU。WINDOWS95，98。这种OS可能会出现某个进程长期独占CPU的情况，如死循环，这样会影响其他进程执行的公平和及时。
- 内核完全不可抢占：在用户态下执行代码可以随时被抢用CPU，但在核心态下执行代码，则完全不可以被抢用CPU。例如传统的UNIX（SVR3和4.3BSD UNIX及其以前的版本）、WINDOWS NT。这些OS通常在系统调用或中断处理时屏蔽大部分中断，系统调用返回或中断返回时再开放大部分中断。

优先权调度算法(Priority Scheduling)

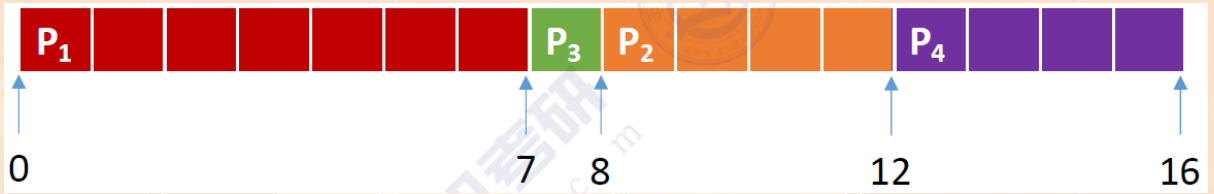
- 内核部分可抢占：在用户态下执行代码可以随时被抢用CPU，但在核心态时则大部分时间都不可以被抢用CPU，而只在某些时刻（称为可抢占点，Preemption Point），可以被抢用CPU。例如 UNIX SVR4。
- 完全可抢占或内核完全可抢占：无论处于用户态还是核心态，都可以随时被抢用CPU。例如：Solaris 、Windows 2000 / XP。实际上，Solaris和Windows 2000 / XP并不是100%完全可抢先，它只是将内核中不可抢先的代码段尽量减少而已。任何OS都不可能是100%的完全可抢先的。

优先权调度算法(Priority Scheduling)

例题：各进程到达就绪队列的时间、需要运行的时间、进程优先数如下表所示。使用非抢占式的优先级调度算法，分析进程运行情况。使用非抢占式的优先级调度算法，分析进程运行情况。（注：优先数越大，优先级越高）

进程	到达时间	运行时间	优先数
P1	0	7	1
P2	2	4	2
P3	4	1	3
P4	5	4	2

非抢占式的优先级调度算法：每次调度时选择当前已到达且优先级最高的进程。当前进程主动放弃处理机时发生调度。



注：以下括号内表示当前处于就绪队列的进程。

0时刻(P1): 只有P1到达，P1上处理机。

7时刻(P2、P3、P4): P1运行完成主动放弃处理机，其余进程都已到达，P3优先级最高，P3上处理机。

8时刻(P2、P4): P3完成，P2、P4优先级相同，由于P2先到达，因此P2优先上处理机

12时刻(P4): P2完成，就绪队列只剩P4，P4上处理机。

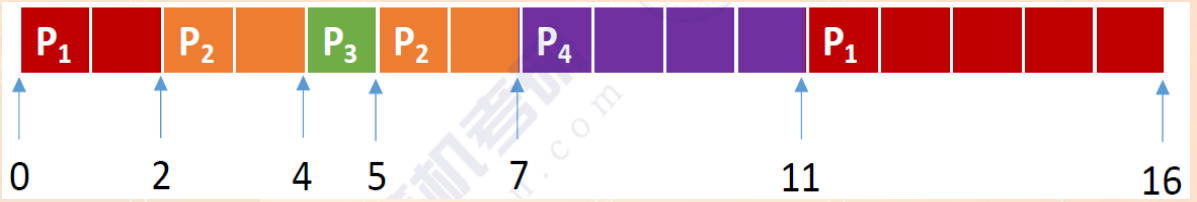
16时刻(): P4完成，所有进程都结束。

优先权调度算法(Priority Scheduling)

例题：各进程到达就绪队列的时间、需要运行的时间、进程优先数如下表所示。使用非抢占式的优先级调度算法，分析进程运行情况。使用抢占式的优先级调度算法，分析进程运行情况。（注：优先数越大，优先级越高）

进程	到达时间	运行时间	优先数
P1	0	7	1
P2	2	4	2
P3	4	1	3
P4	5	4	2

抢占式的优先级调度算法：每次调度时选择当前已到达且优先级最高的进程。当前进程主动放弃处理机时发生调度。另外，当就绪队列发生改变时也需要检查是否会发生抢占。



注：以下括号内表示当前处于就绪队列的进程。

0时刻(P1): 只有P1到达，P1上处理机。

2时刻(P2): P2到达就绪队列，优先级比P1更高，发生抢占。P1回就绪队列，P2上处理机。

4时刻(P1、P3): P3到达，优先级比P2更高，P2回到就绪队列，P3抢占处理机。

5时刻(P1、P2、P4): P3完成，主动释放处理机，同时，P4也到达，但是，由于P2比P4更先进入就绪队列，因此优先选择P2上处理机。

7时刻(P1、P4): P2完成，就绪队列只剩P1、P4，再次调度，P4上处理机(进程优先数)。

11时刻(P1): P4完成，P1上处理机。

16时刻(): P1完成，所有进程均完成。

优先权调度算法(Priority Scheduling)

补充：

就绪队列未必只有一个，可以按照不同优先级来组织。另外，也可以把优先级高的进程排在更靠近队头的位置。

根据优先级是否可以动态改变，可将优先级分为静态优先级程动态优先级两种

静态优先级：创建进程时确定，之后一直不变。

动态优先级：创建进程时有一个初始值，之后会根据情况

I/O设备和CPU可以并行工作。如果优先让I/O繁忙型进程优先运行的话，则越有可能让I/O设备尽早地投入工作，则资源利用率、系统吞吐量都会得到提升

如何合理地设置各类进程的优先级

通常：系统进程的优先级 高于 用户进

前台进程的优先级 高于 后台进程

操作系统更偏好I/O型进程(或称I/O繁忙型进程)

注：与I/O型进程相对的是计算型进程(或称CPU繁忙型进程)

如何采用的是动态优先级，什么时候应该调整？

可以从追求公平、提升资源利用率等角度考虑

如果某进程在就绪队列中等待了很长时间，则可以适当提升其优先级

如果某进程占用处理机运行了很长时间，则可以适当降低其优先级

如果发现一个进程频繁地进行I/O操作，则可适当提升其优先级

优先权的类型

- 静态优先级
  - 创建进程时就确定，直到进程终止前都不改变。通常是一个整数。
  - 依据：
    - 进程类型（系统进程优先级较高）
    - 对资源的需求（对CPU和内存需求较少的进程，优先级较高）
    - 用户要求（紧迫程度和付费多少）
  - 特点：
    - 简单，系统开销小
    - 不精确，仅在要求不高的系统中使用

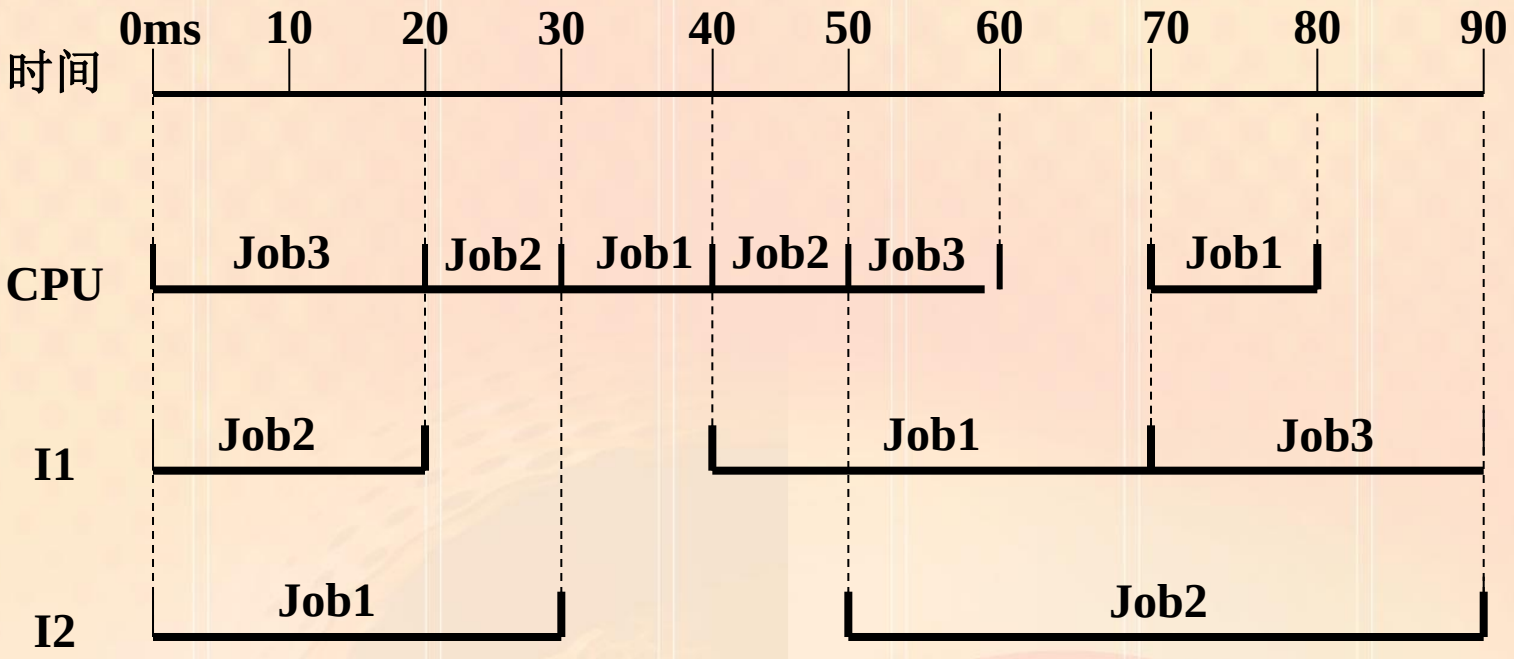
优先权的类型

- 动态优先级
  - 在创建进程时赋予的优先级，在进程运行过程中可以自动改变，以便获得更好的调度性能。
  - 如：
    - 在就绪队列中，等待时间延长则优先级提高，从而使优先级较低的进程在等待足够的时间后，其优先级提高到可被调度执行；
    - 进程每执行一个时间片，就降低其优先级，从而一个进程持续执行时，其优先级降低到出让CPU。

优先权调度算法(Priority Scheduling)

- 例题：在单CPU和两台I/O（I1,I2)设备的多道程序环境下，同时投入三个作业运行，它们的执行轨迹如下：
  - Job1: I2(30ms)　CPU(10ms)　I1（30ms)　CPU（10ms)
  - Job2: I1（20ms)　CPU(20ms)　I2(40ms)
  - Job3: CPU(30ms)　I1(20ms)
- 如果CPU、I1、I2都能并行工作，优先级从高到低为Job1、Job2、Job3，采用可抢占式优先级调度方式。
- 求(1) 每个作业的周转时间(2) CPU的利用率(3) I/O设备利用率

优先权调度算法(Priority Scheduling)



Job1的周转时间80ms，Job2 和Job3周转时间各为90ms

CPU利用率＝ 70ms/90ms = 77.78%

I1 利用率＝ I2利用率 = 70ms /90ms ＝ 77.78%

优先权调度算法(Priority Scheduling)

优先级调度

算法思想	随着计算机的发展，特别是实时操作系统的出现，越来越多的应用场景需要根据任务的紧急程度来决定处理顺序
算法规则	调度时选择优先级最高的作业/进程
用于作业/进程调度	既可用于作业调度，也可用于进程调度。甚至，还会用于在之后会学习的I/O调度中
是否可抢占？	抢占式、非抢占式都有。做题时的区别在于：非抢占式只需在进程主动放弃处理机时进行调度即可，而抢占式还需在就绪队列变化时，检查是否会发生抢占。
优缺点	优点：用优先级区分紧急程度、重要程度，适用于实时操作系统。可灵活地调整对各种作业/进程的偏好程度。 缺点：若源源不断地有高优先级进程到来，则可能导致饥饿
是否会导致饥饿	会

思考

FCFS算法的优点是公平

SJF 算法的优点是能尽快处理完短作业，平均等待/周转时间等参数很优秀

时间片轮转调度算法可以让各个进程得到及时的响应

优先级调度算法可以灵活地调整各种进程被服务的机会

能否对其他算法做个折中权衡？得到一个综合表现优秀平衡的算法呢？

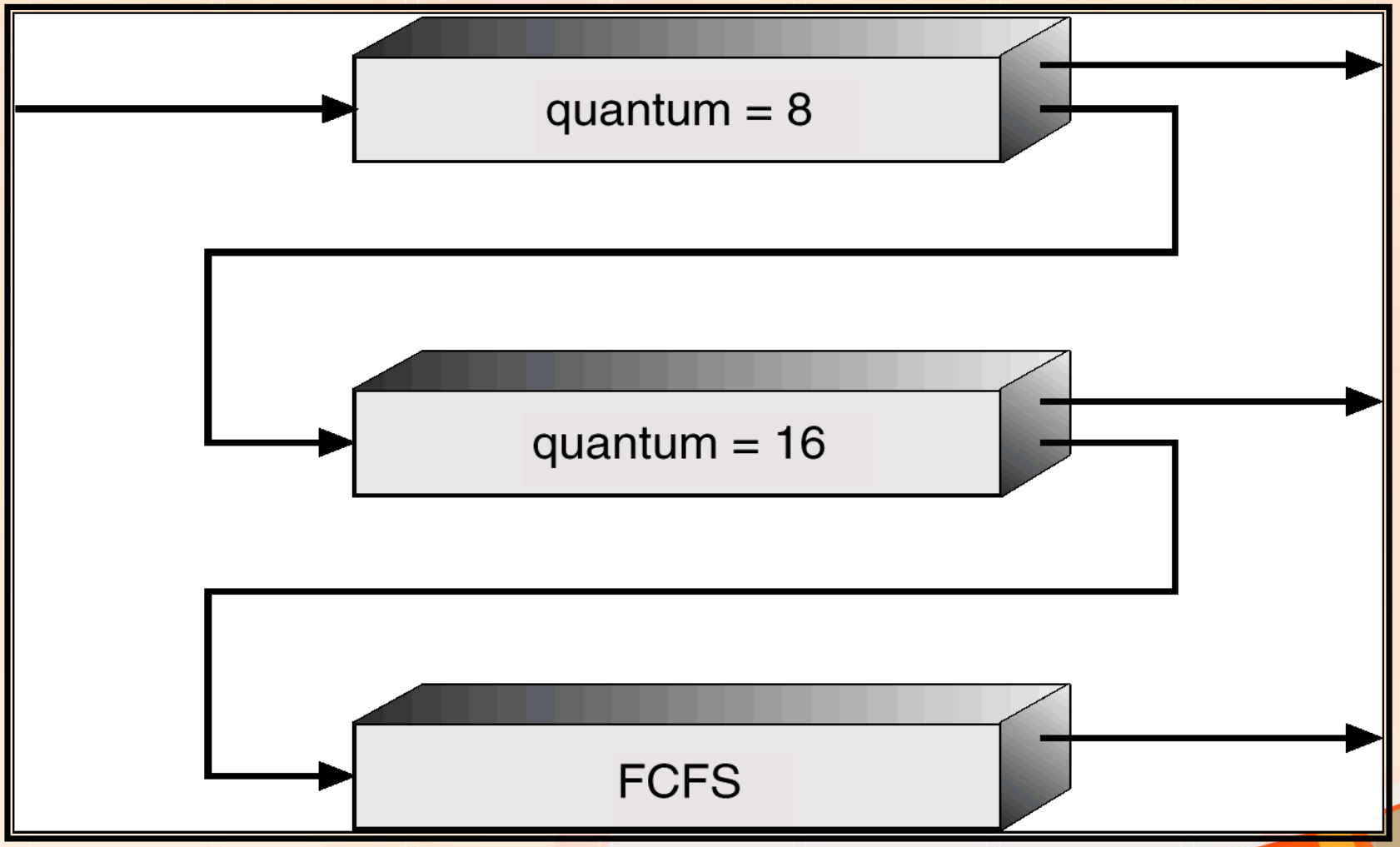
厉害了，我的哥

多级反馈队列调度算法

多级反馈队列调度算法(Round Robin with Multiple Feedback)

- 多级反馈队列算法是时间片轮转算法和优先级算法的综合和发展。
- 1) 算法描述
  - 设置多个就绪队列，分别赋予不同的优先级，队列1的优先级最高。每个队列执行时间片的长度也不同，规定优先级越低则时间片越长。
  - 假设有三个就绪队列：
    - Q1——时间片为8
    - Q2——时间片为16
    - Q3——FCFS
  - 新进程进入内存后，先投入队列1的末尾，若按队列1一个时间片未能执行完，则降低投入到队列2的末尾，若仍未完成，降低到最后的队列，按FCFS算法调度直到完成。
  - 仅当较高优先级的队列为空，才调度较低优先级的队列中的进程执行。如果进程执行时有新进程进入较高优先级的队列，则抢先执行新进程，并把被抢先的进程投入原队列的末尾。

多级反馈队列调度算法



- 2）算法性能
  - 终端型进程：让其进入最高优先级队列，以及时响应I/O交互。通常执行一个小时间片，可处理完一次I/O请求的数据，然后转入到阻塞队列。
  - 计算型进程（长批处理作业）：每次都执行完时间片，进入更低级队列。最终采用最大时间片来执行，减少调度次数。
  - 短批处理作业：先放入第1级，一般经过1，2级即可完成。

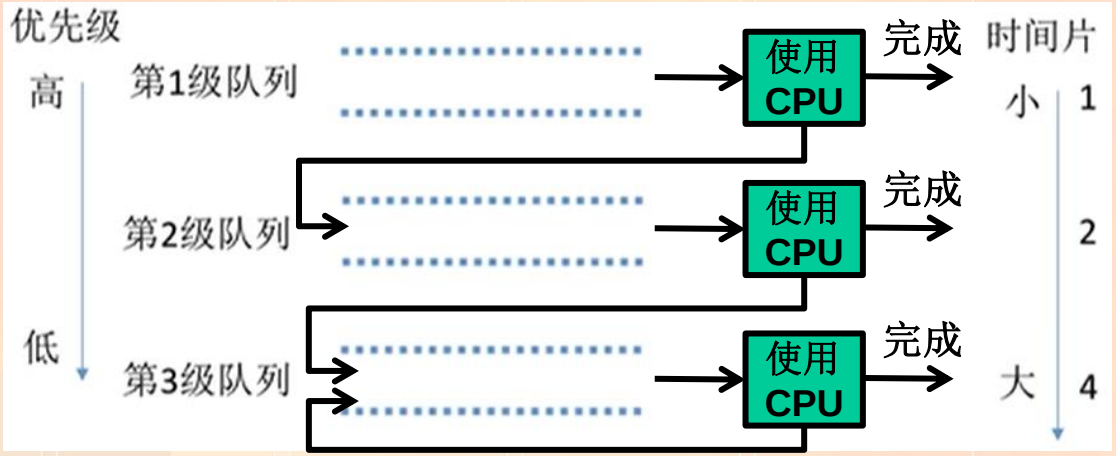
多级反馈队列调度算法

- 3）优点：
  - 为提高系统吞吐量和缩短平均周转时间而照顾短进程
  - 为获得较好的I/O设备利用率和缩短响应时间而照顾I/O型进程
  - 不必估计进程的执行时间，动态调节

多级反馈队列调度算法(Round Robin with Multiple Feedback)

例题: 各进程到达就绪队列的时间，需要的运行时间如下表所示。使用多级反馈队列调度算法，分析进程运行的过程。

进程	到达时间	运行时间
P1	0	8
P2	1	4
P3	5	1



P1(1) ==> P2(1)==>P1(2)==>
P2(1)==>P3(1)==>P2(2) ==>
P1(4)==>P1(1)

设置多级就绪队列，各级队列优先级从高到低，时间片从小到大。
新进程到达时先进入第1级队列，按FCFS原则排队等待被分割的时间片。若用完时间片进程还未结束，则进程进入下一级队列队尾。如果此时已经在最下级的队列，则重新放回最下级队列队尾。
只有第k级队列为空时，才会为k+1级队头的进程分配时间片。
被抢占处理机的进程重新放回原队列队尾。

多级反馈队列调度算法(Round Robin with Multiple Feedback)

多级反馈队列

算法思想	对其他调度算法的折中权衡
算法规则	1. 设置多级就绪队列，各级队列优先级从高到低，时间片从小到大 2. 新进程到达时先进入第1级队列，按FCFS原则排队等待被分配时间片，若用完时间片进程还未结束，则进程进入下一级队列队尾。如果此时已经是在最下级的队列，则重新放回该队列队尾 3. 只有第 k 级队列为空时，才会为 k+1 级队头的进程分配时间片
用于作业/进程调度	用于进程调度
是否可抢占?	抢占式的算法。在 k 级队列的进程运行过程中，若更上级的队列（1~k-1级）中进入了一个新进程，则由于新进程处于优先级更高的队列中，因此新进程会抢占处理机，原来运行的进程放回 k 级队列队尾。
优缺点	对各类型进程相对公平（FCFS的优点）；每个新到达的进程都可以很快就得到响应（RR的优点）；短进程只用较少的时间就可完成（SPF的优点）；不必实现估计进程的运行时间（避免用户作假）；可灵活地调整对各类进程的偏好程度，比如CPU密集型进程、I/O密集型进程（拓展：可以将因I/O而阻塞的进程重新放回原队列，这样I/O型进程就可以保持较高优先级）
是否会导致饥饿	会

总结

算法	思想&规则	可抢占？	优点	缺点	会导致饥饿？	补充
时间片轮转		抢占式	公平，适用于分时系统	频繁切换有开销，不区分优先级	不会	时间片太大或太小有何影响？
优先级调度		有抢占式的，也有非抢占式的。注意做题时的区别	区分优先级，适用于实时系统	可能导致饥饿	会	动态/静态优先级。 各类型进程如何设置优先级？如何调整优先级？
多级反馈队列	较复杂，注意理解	抢占式	平衡优秀 666	一般不说它有缺点，不过可能导致饥饿	会	

注：比起早期的批处理操作系统来说，由于计算机造价大幅度降低，因此之后出现的交互式操作系统(包括分时操作系统、实时操作系统等)更注重系统的响应时间、公平性、平衡性等指标。而这几种算法恰好也能较好地满足交互式系统的需求。因此三种算法适合用于交互式系统（比如UNIX使用的就是多级反馈队列调度算法）。

多级队列调度算法

- 系统中按照进程类型设置多个队列，进程创建成功后插入某个队列

队列之间可采取固定优先级，或时间片划分
固定优先级：高优先级空时，低优先级进程才能被调度
时间片划分：如三个队列分配时间50%，40%，10%

各队列采用不同的调度策略，如：系统进程队列采用优先级调度；交互式队列采用RR，批处理队列采用FCFS

- 例：假设某操作系统采用时间片轮转调度策略，时间片大小为100ms，就绪进程队列的平均长度为5，如果在系统中运行一个需要在CPU上执行0.8s时间的程序，则该程序的平均周转时间是_____，平均等待时间是_____。（不考虑IO情况及系统调度开销）

  - 答： 排在队列尾的周转时间4s，排在队列第四的周转时间3.9s.....排在头的周转时间3.6s，平均：3.8s
  - 同理：平均等待 3.0s

作业

进程	到达时间	执行时间
P1	0	10
P2	1	8
P3	2	2
P4	3	4
P5	4	8

画Gantt图说明使用FCFS、HRRN、RR（时间片＝3）调度算法进程调度情况。并分别求这几种算法的平均周转时间，平均等待时间。

作业

有一个内存中只能装入两道作业的批处理系统。作业调度采用最高优先权算法，进程调度采用抢占式短作业优先算法。表给出了4个进程的到达时间、要求运行时间和优先权，其中数字越小优先权越高。

1）请将表格填写完整；

2）计算系统的平均带权周转时间。

进程	提交时间	运行时间	优先权	开始执行时间	完成时间
P1	2:00	3小时	5		
P2	0:00	5小时	3		
P3	0:00	1小时	4		
P4	2:00	2小时	6		

作业

- 若主存中有3道程序A、B、C，它们按A、B、C优先次序运行，采用可抢占式优先级调度方式。各程序的计算轨迹为：

  A：计算（20）、I/O(30)、计算（10）

  B：计算（40）、I/O(20)、计算（10）

  C：计算（10）、I/O(30)、计算（20）

- 如果三道程序都使用相同设备运行I/O（即程序用串行方式使用设备，调度开销忽略不计），分别画出单道和多道运行的时间关系图，并计算两种情况下，CPU的平均利用率。