

本节内容

调度算法

时间片轮转
优先级调度
多级反馈队列

时间片轮转调度算法 (RR)



优先级调度算法

多级反馈队列调度算法

Tips: 各种调度算法的学习思路

1. 算法思想
2. 算法规则
3. 这种调度算法是用于 作业调度 还是 进程调度？
4. 抢占式？非抢占式？
5. 优点和缺点
6. 是否会导致 **饥饿**

某进程/作业长期
得不到服务

时间片轮转 (RR, Round-Robin)

时间片轮转

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

是否会导致饥饿

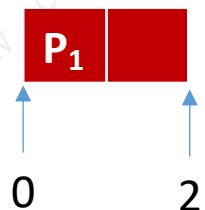
时间片轮转 (RR, Round-Robin)

常用于分时操作系统，更注重“响应时间”，因此此处不计算周转时间

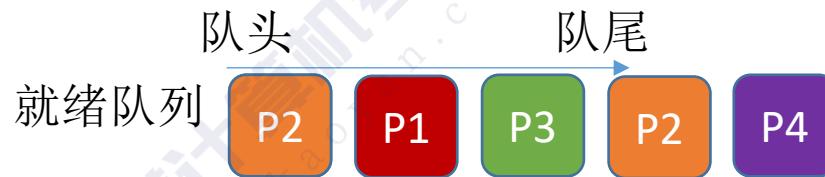
例题：各进程到达就绪队列的时间、需要的运行时间如下表所示。使用时间片轮转调度算法，分析时间片大小分别是2、5时的进程运行情况。

进程	到达时间	运行时间
P1	0	5
P2	2	4
P3	4	1
P4	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、进程的剩余运行时间)



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，重新放到队尾。此时P2排在队头，因此让P2上处理机。（注意：2时刻，P1下处理机，同一时刻新进程P2到达，如果在题目中遇到这种情况，默认新到达的进程先进入就绪队列）

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，P2下处理机也插到队尾

5时刻 (P3(1) → P2(2) → P4(6)) :5时刻，P4到达插到就绪队尾（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）

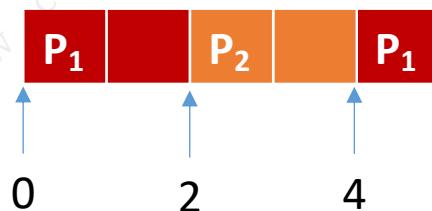
时间片轮转 (RR, Round-Robin)

常用于分时操作系统，更注重“响应时间”，因此此处不计算周转时间

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
P1	0	5
P2	2	4
P3	4	1
P4	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、进程的剩余运行时间)

6时刻 ($P3(1) \rightarrow P2(2) \rightarrow P4(6) \rightarrow P1(1)$)：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 ($P2(2) \rightarrow P4(6) \rightarrow P1(1)$)：虽然P3的时间片没用完，但是由于P3只需运行1个单位的时间，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 ($P4(6) \rightarrow P1(1)$)：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 ($P1(1) \rightarrow P4(4)$)：P4时间片用完，重新回到就绪队列。P1上处理机



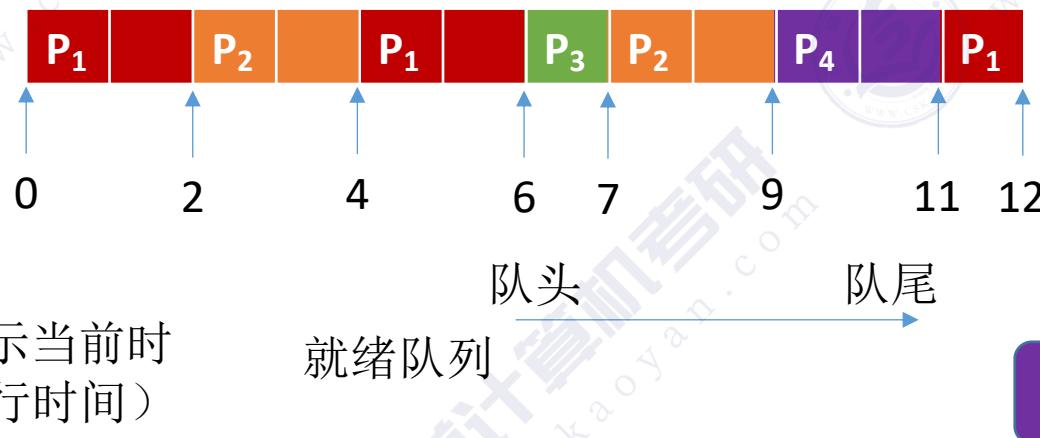
时间片轮转 (RR, Round-Robin)

常用于分时操作系统，更注重“响应时间”，因此此处不计算周转时间

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
P1	0	5
P2	2	4
P3	4	1
P4	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、进程的剩余运行时间)

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 ()：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

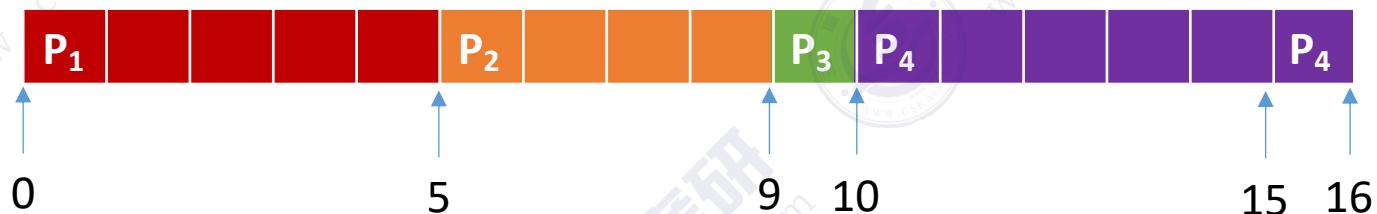
时间片轮转 (RR, Round-Robin)

常用于分时操作系统，更注重“响应时间”，因此此处不计算周转时间

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
P1	0	5
P2	2	4
P3	4	1
P4	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 5

0时刻 (P1(5))：只有P1到达，P1上处理机。

2时刻 (P2(4))：P2到达，但P1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。

16时刻 ()：P4运行完，主动放弃处理机。所有进程运行完。

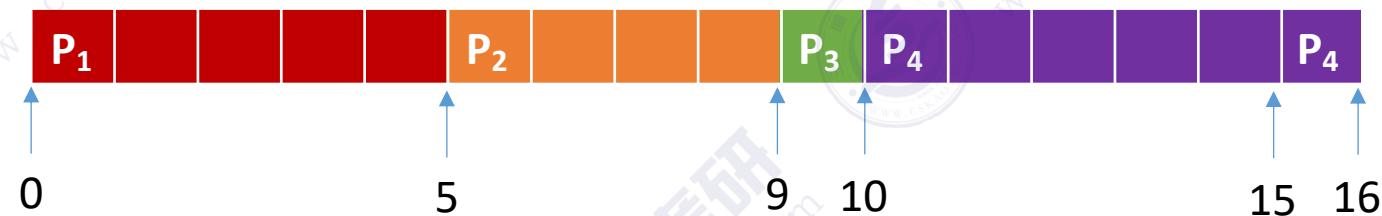
时间片轮转 (RR, Round-Robin)

常用于分时操作系统，更注重“响应时间”，因此此处不计算周转时间

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
P1	0	5
P2	2	4
P3	4	1
P4	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，使得每个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。

时间片轮转 (RR, Round-Robin)

常用于分时操作系统，更注重“响应时间”，因此此处不计算周转时间

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
P1	0	5
P2	2	4
P3	4	1
P4	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，使得每个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。

时间片轮转 (RR, Round-Robin)

常用于分时操作系统，更注重“响应时间”，因此此处不计算周转时间

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
P1	0	5
P2	2	4
P3	4	1
P4	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...

一般来说，设计时间片时要让切换进程的开销占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，比不超过1%大大进程响应时间。因此时间片不能太大。

另一方面，进程调度的直接代价的（保存、恢复运行环境）。因此如果时间片太小，会导致进程执行的时间比

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 (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？

优先级调度算法

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

每个作业/进程有各自的优先级，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

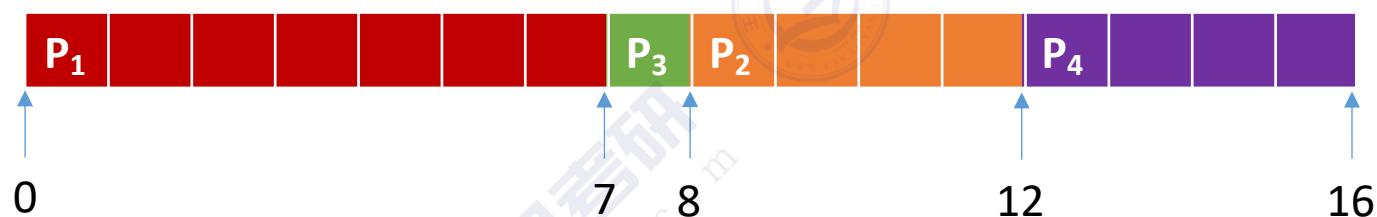
是否会导致饥饿

优先级调度算法

例题：各进程到达就绪队列的时间、需要的运行时间、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。（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。

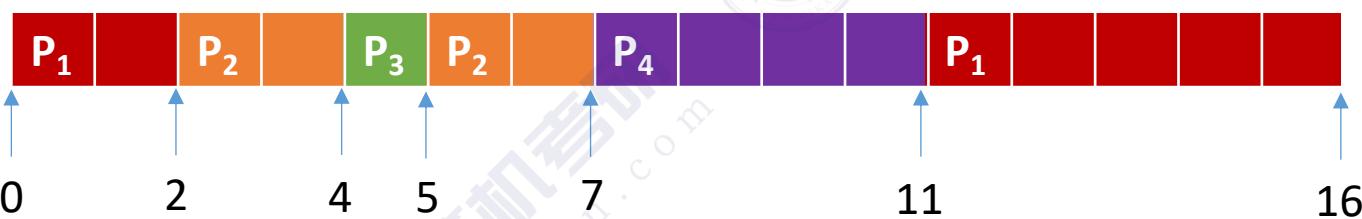
16时刻（）：P4完成，所有进程都结束

优先级调度算法

例题：各进程到达就绪队列的时间、需要的运行时间、进程优先数如下表所示。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回到就绪队列，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，由于P2比P4更先进入就绪队列，因此选择P2上处理机

7时刻（P1、P4）：P2完成，就绪队列只剩P1、P4，P4上处理机。

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

优先级调度算法

补充：

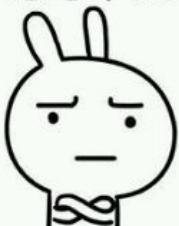
就绪队列未必只有一个，可以按照不同优先级来组织。另外，也可以把优先级高的进程排在更靠近队头的位置

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和动态优先级两种。

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，之后会根据情况动态地调整优先级。

思考中.....



如何合理地设置各类进程的优先级？

思考中.....



如果采用的是动态优先级，什么时候应该调整？

通常： 系统进程优先级 **高于** 用户进程
前台进程优先级 **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）

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

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

优先级调度算法

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...

思考中.....



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？

多级反馈队列调度算法

厉害了，我的哥



多级反馈队列调度算法

多级
反馈
队列

算法思想

算法规则

用于作业/进程调度

是否可抢占?

优缺点

是否会导致饥饿

对其他调度算法的折中权衡

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

用于进程调度

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（ $1 \sim k-1$ 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。

多级反馈队列调度算法

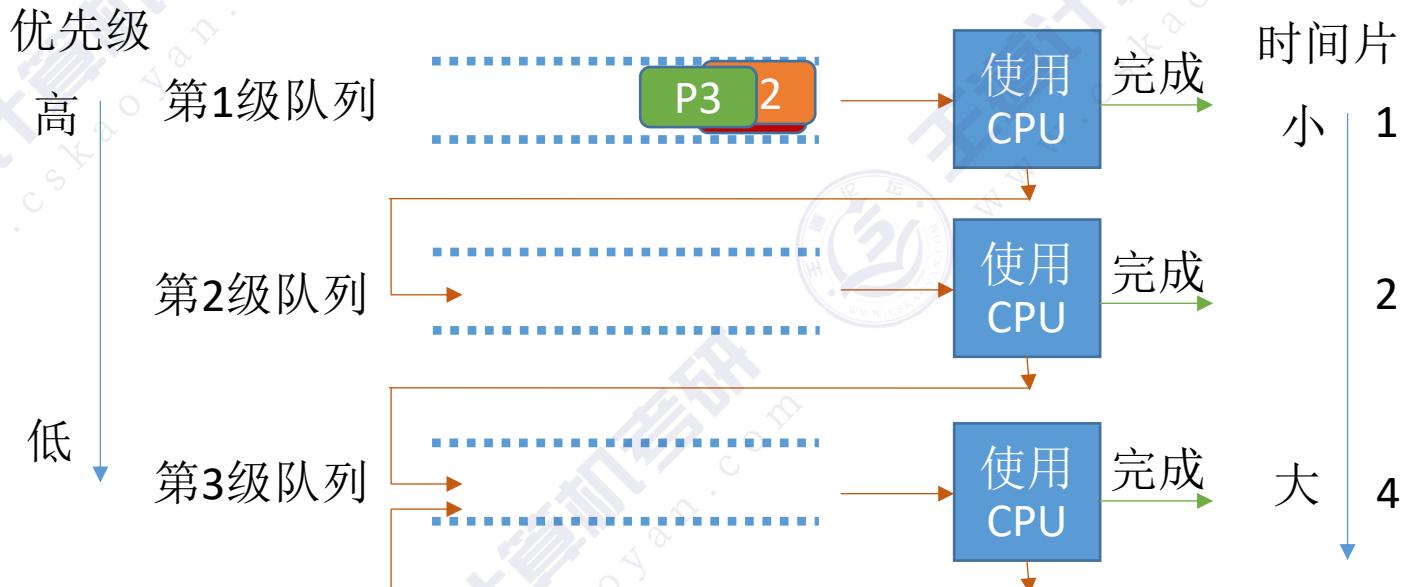
例题：各进程到达就绪队列的时间、需要的运行时间如下表所示。使用**多级反馈队列**调度算法，分析进程运行的过程。

进程	到达时间	运行时间
P1	0	8
P2	1	4
P3	5	1

$P1(1) \rightarrow P2(1) \rightarrow P1(2)$

$\rightarrow P2(1) \rightarrow P3(1) \rightarrow P2(2)$

$\rightarrow P1(4) \rightarrow P1(1)$



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

多级反馈队列

算法思想

算法规则

用于作业/进程调度

是否可抢占？

优缺点

是否会导致饥饿

对其他调度算法的折中权衡

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

用于进程调度

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（ $1 \sim k-1$ 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。

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）

会

知识回顾与重要考点

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。因此这三种算法适合用于**交互式系统**。(比如UNIX使用的就是多级反馈队列调度算法)

提示: 一定要动手做课后习题!