

典型的调度算法

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

先来先服务 (FCFS)

FCFS, First Come First Serve

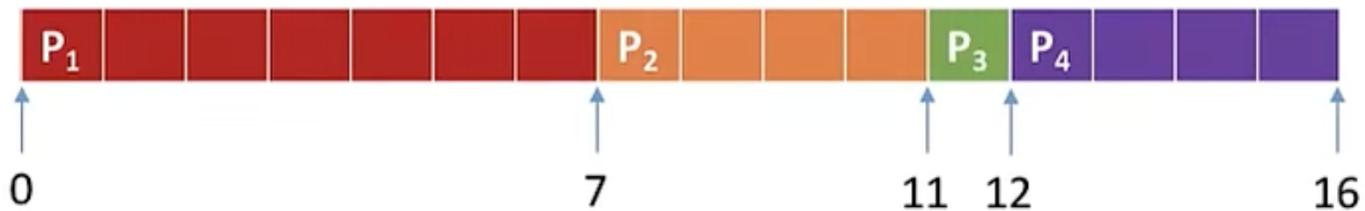
先来先服务	
算法思想	主要从“公平”的角度考虑（类似于我们生活中排队买东西的例子）
算法规则	按照作业/进程到达的先后顺序进行服务
用于作业/进程调度	用于作业调度时，考虑的是哪个作业先到达后备队列 用于进程调度时，考虑的是哪个进程先到达就绪队列
是否可抢占？	非抢占式的算法
优点	公平、算法实现简单
缺点	排在长作业（进程）后面的短作业需要等待很长时间，带权周转时间很大，对短作业来说用户体验不好。即，FCFS算法对长作业有利，对短作业不利
是否会导致饥饿	不会

各进程到达就绪队列的时间、需要的运行时间如下表所示。使用先来先服务调度算法，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

先来先服务调度算法：按照到达的先后顺序调度，事实上就是等待时间越久的越优先得到服务。

调度顺序为： $P1 \rightarrow P2 \rightarrow P3 \rightarrow P4$



周转时间

$$\text{周转时间} = \text{完成时间} - \text{到达时间} \quad (1)$$

$$P1 = 7 - 0 = 7 \quad (2)$$

$$P2 = 11 - 2 = 9 \quad (3)$$

$$P3 = 12 - 4 = 8 \quad (4)$$

$$P4 = 16 - 5 = 11 \quad (5)$$

注意：本例中的进程都是纯计算型的进程，一个进程到达后要么在等待，要么在运行。如果是又有计算、又有I/O操作的进程，其等待时间就是

$$\text{等待时间} = \text{周转时间} - \text{运行时间} - I/O\text{操作的时间} \quad (6)$$

带权周转时间

$$\text{带权周转时间} = \text{周转时间}/\text{运行时间} \quad (7)$$

$$P1 = 7/7 = 1 \quad (8)$$

$$P2 = 9/4 = 2.25 \quad (9)$$

$$P3 = 8/1 = 8 \quad (10)$$

$$P4 = 11/4 = 2.75 \quad (11)$$

等待时间

$$\text{等待时间} = \text{周转时间} - \text{运行时间} \quad (12)$$

$$P1 = 7 - 7 = 0 \quad (13)$$

$$P2 = 9 - 4 = 5 \quad (14)$$

$$P3 = 8 - 1 = 7 \quad (15)$$

$$P4 = 11 - 4 = 7 \quad (16)$$

平均周转时间

$$\text{平均周转时间} = (7 + 9 + 8 + 11)/4 = 8.75 \quad (17)$$

平均带权周转时间

$$\text{平均带权周转时间} = (1 + 2.25 + 8 + 2.75)/4 = 3.5 \quad (18)$$

平均等待时间

$$\text{平均等待时间} = (0 + 5 + 7 + 7)/4 = 4.75 \quad (19)$$

短作业优先 (SJF)

注意几个细节：

1. 如果题目中未特别说明，所提到的“短作业/短进程优先算法”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（FCFS），SJF依然可以获得较少的平均等待时间、平均周转时间
4. 如果遇到“SJF算法的平均等待时间、平均周转时间最少”，比较其他选项。

SJF, Shortest Job First

短作业优先	
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。 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”

非抢占式

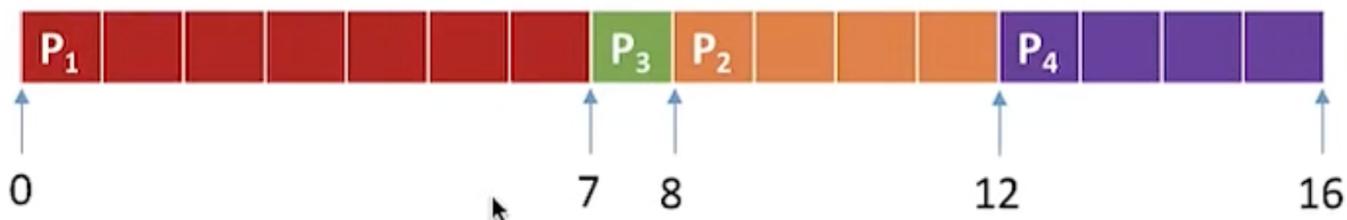
各进程到达就绪队列的时间、需要的运行时间如下表所示。使用非抢占式的短作业优先调度算法，计算各进程的等待时间、平均等待时间、周转时间、平均周转时间、带权周转时间、平均带权周转时间。

严格来说，用于进程调度应该称为短进程优先调度算法 (SPF)

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

短作业/进程优先调度算法：每次调度时选择当前已到达且运行时间最短的作业/进程。

调度顺序为：P1->P3->P2->P4



周转时间

$$\text{周转时间} = \text{完成时间} - \text{到达时间} \quad (20)$$

$$P1 = 7 - 0 = 7 \quad (21)$$

$$P3 = 8 - 4 = 4 \quad (22)$$

$$P2 = 12 - 2 = 10 \quad (23)$$

$$P4 = 16 - 5 = 11 \quad (24)$$

注意：本例中的进程都是纯计算型的进程，一个进程到达后要么在等待，要么在运行。如果是又有计算、又有I/O操作的进程，其等待时间就是

$$\text{等待时间} = \text{周转时间} - \text{运行时间} - I/O\text{操作的时间} \quad (25)$$

带权周转时间

$$\text{带权周转时间} = \text{周转时间}/\text{运行时间} \quad (26)$$

$$P1 = 7/7 = 1 \quad (27)$$

$$P3 = 4/1 = 4 \quad (28)$$

$$P2 = 10/4 = 2.5 \quad (29)$$

$$P4 = 11/4 = 2.75 \quad (30)$$

等待时间

$$\text{等待时间} = \text{周转时间} - \text{运行时间} \quad (31)$$

$$P1 = 7 - 7 = 0 \quad (32)$$

$$P3 = 4 - 1 = 3 \quad (33)$$

$$P2 = 10 - 4 = 6 \quad (34)$$

$$P4 = 11 - 4 = 7 \quad (35)$$

平均周转时间

$$\text{平均周转时间} = (7 + 4 + 10 + 11)/4 = 8 \quad (36)$$

$$FCFS = 8.75 \quad (37)$$

平均带权周转时间

$$\text{平均带权周转时间} = (1 + 4 + 2.5 + 2.75)/4 = 2.56 \quad (38)$$

$$FCFS = 3.5 \quad (39)$$

平均等待时间

$$\text{平均等待时间} = (0 + 3 + 6 + 7)/4 = 4 \quad (40)$$

$$FCFS = 4.75 \quad (41)$$

对比FCFS算法的结果，显SPF算法的平均等待/周转/带权周转时间都要更低

抢占式

各进程到达就绪队列的时间、需要的运行时间如下表所示。使用抢占式的短作业优先调度算法，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。

抢占式的短作业优先算法又称“最短剩余时间优先算法（SRTN）”

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

最短剩余时间优先算法：每当有进程加入就绪队列改变时就需要调度，如果新到达的进程剩余时间比当前运行的进程剩余时间更短，则由新进程抢占处理机，当前运行进程重新回到就绪队列。另外，当一个进程完成时也需要调度

需要注意的是，当有新进程到达时就绪队列就会改变，就要按照上述规则进行检查。以下Pn(m)表示当前Pn进程剩余时间为m。各个时刻的情况如下：

0时刻到达（P1到达）：P1(7)

2时刻到达（P2到达）：P1(5)、P2(4)

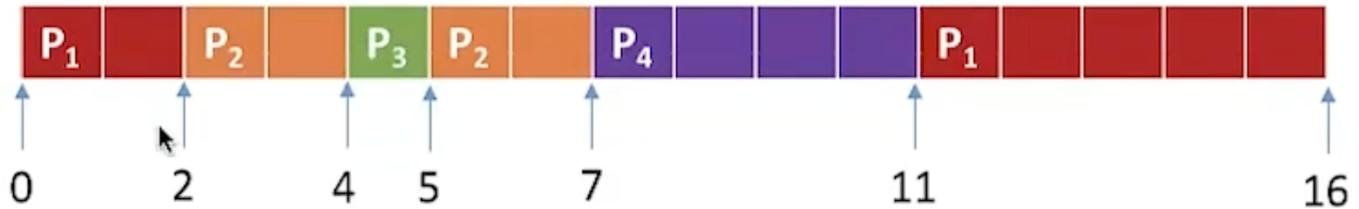
4时刻到达（P3到达）：P1(5)、P2(2)、P3(1)

5时刻到达（P3完成且P4刚好到达）：P1(5)、P2(2)、P4(4)

7时刻到达（P2完成）：P1(5)、P4(4)

11时刻到达（P4完成）：P1(5)

调度顺序为：P1->P2->P3->P2->P4->P1



周转时间

$$\text{周转时间} = \text{完成时间} - \text{到达时间} \quad (42)$$

$$P1 = 16 - 0 = 16 \quad (43)$$

$$P2 = 7 - 2 = 5 \quad (44)$$

$$P3 = 5 - 4 = 1 \quad (45)$$

$$P4 = 11 - 5 = 6 \quad (46)$$

带权周转时间

$$\text{带权周转时间} = \text{周转时间}/\text{运行时间} \quad (47)$$

$$P1 = 16/7 = 1 \quad (48)$$

$$P2 = 5/4 = 1.25 \quad (49)$$

$$P3 = 1/1 = 1 \quad (50)$$

$$P4 = 6/4 = 1.5 \quad (51)$$

等待时间

$$\text{等待时间} = \text{周转时间} - \text{运行时间} \quad (52)$$

$$P1 = 16 - 7 = 9 \quad (53)$$

$$P2 = 5 - 4 = 1 \quad (54)$$

$$P3 = 1 - 1 = 0 \quad (55)$$

$$P4 = 6 - 4 = 2 \quad (56)$$

平均周转时间

$$\text{平均周转时间} = (16 + 5 + 1 + 6)/4 = 7 \quad (57)$$

$$SJF = 8 \quad (58)$$

平均带权周转时间

$$\text{平均带权周转时间} = (2.28 + 1.25 + 1 + 1.5)/4 = 1.50 \quad (59)$$

$$SJF = 2.56 \quad (60)$$

平均等待时间

$$\text{平均等待时间} = (9 + 1 + 0 + 2)/4 = 3 \quad (61)$$

$$SJF = 4 \quad (62)$$

对比非抢占式的短作业优先算法，显然抢占式的这几个指标又要更低

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

能否设计一个算法，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 (HRRN)

HRRN, Highest Response Ratio Next

$$\text{响应比} = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}} \quad (63)$$

$$\text{响应比} \geq 1 \quad (64)$$

高响应比优先	
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
用于作业/进程调度	既可用于作业调度，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
缺点	\
是否会导致饥饿	不会

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时候（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，选响应比最高的进程上处理机。

$$\text{响应比} = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}} \quad (65)$$

0时刻：只有P1到达就绪队列，P1上处理机

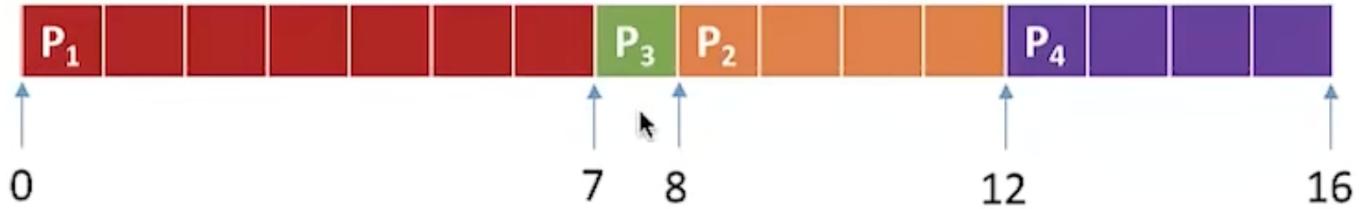
7时刻（P1主动放弃CPU）：就绪队列中有P2（响应比=(5+4)/4=2.25）、P3((3+1)/1=4)、P4((2+4)/4=1.5)

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

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

调度顺序为：P1->P3->P2->P4



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/S PF	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是用于早起的批处理系统，当然，FCFS算法也经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在下方介绍

时间片轮转 (RR)

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？

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。

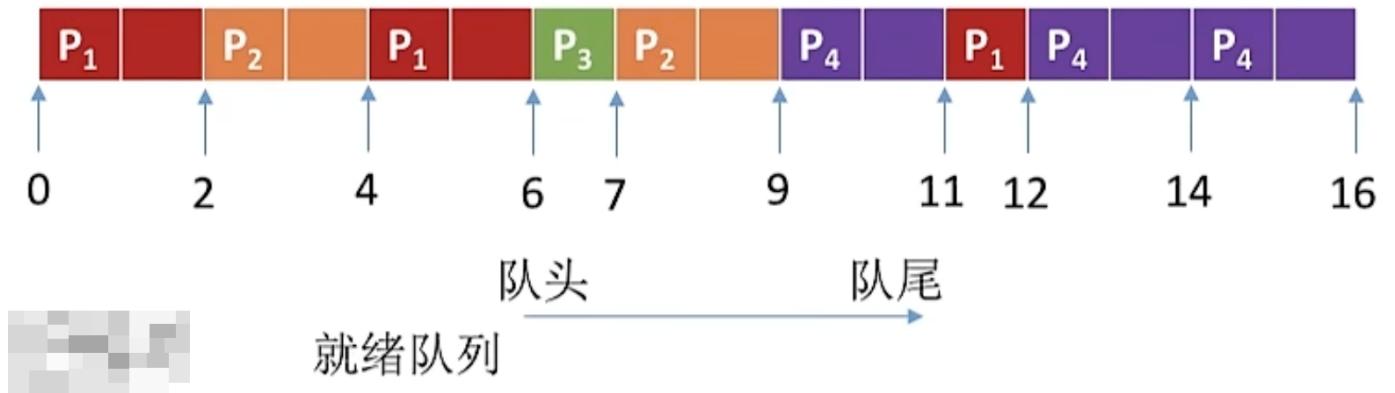
常用于分时操作系统，更注重“响应时间”，因而此处不计算周转时间

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
P1	0	5
P2	2	4
P3	4	1
P4	5	6

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)

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)->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

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

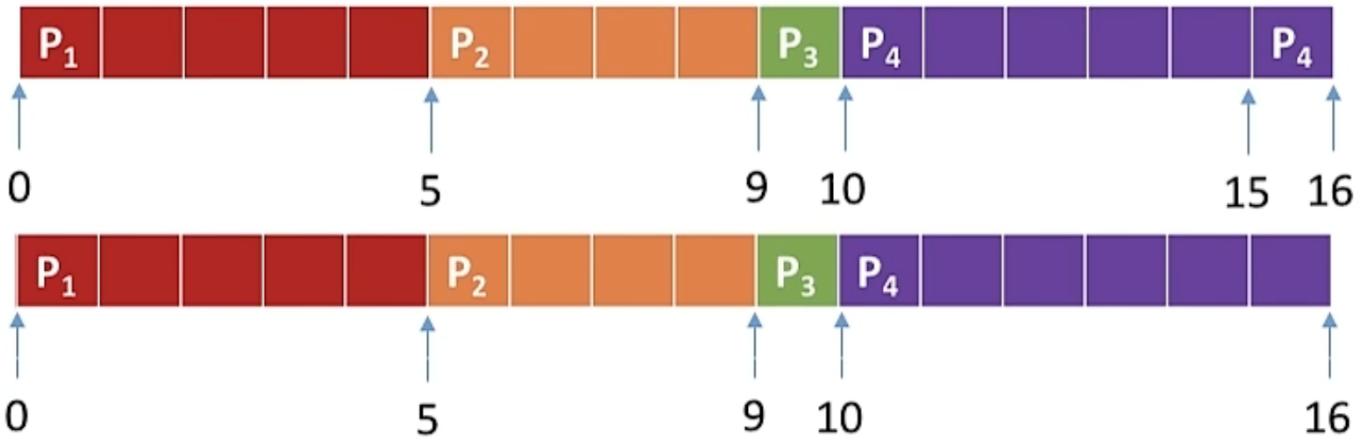
4时刻 (P2(4)->P3(1)) : P3到达, 但P1时间片尚未结束, 因此暂不调度

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。



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。

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9秒才能被系统响应

另一方面，进程调度、切换是有代价的（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%

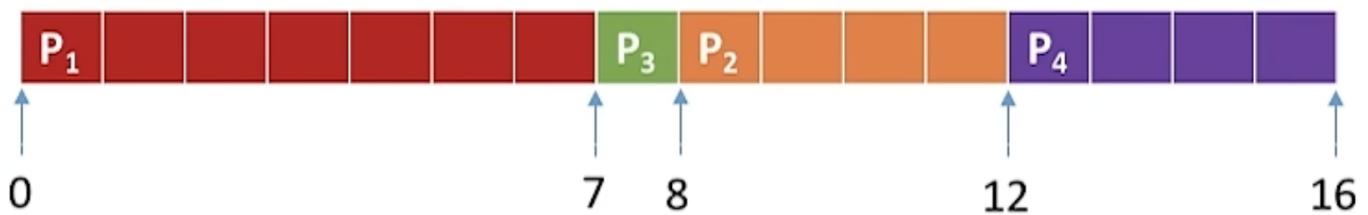
优先级调度

时间片轮转	
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：非抢占式之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	若远远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

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。

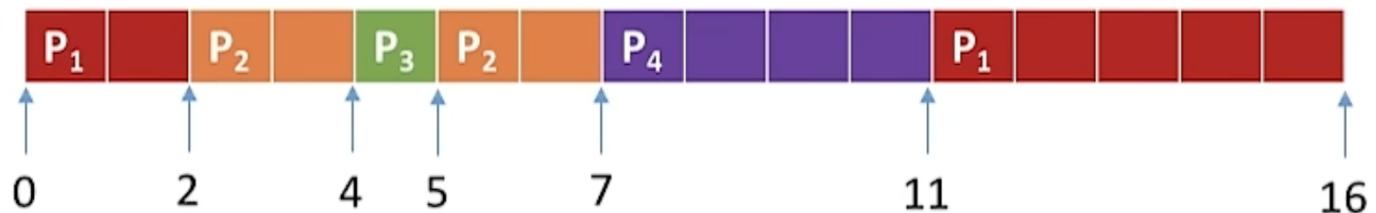
7时刻 (P2、P3、P4) : P1运行完成主动放弃处理机, 其余进程都已到达, P3优先级最高, P3上处理机。

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, 所有进度都结束

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

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。

如何合理地设置各类进程的优先级?

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

如果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

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?

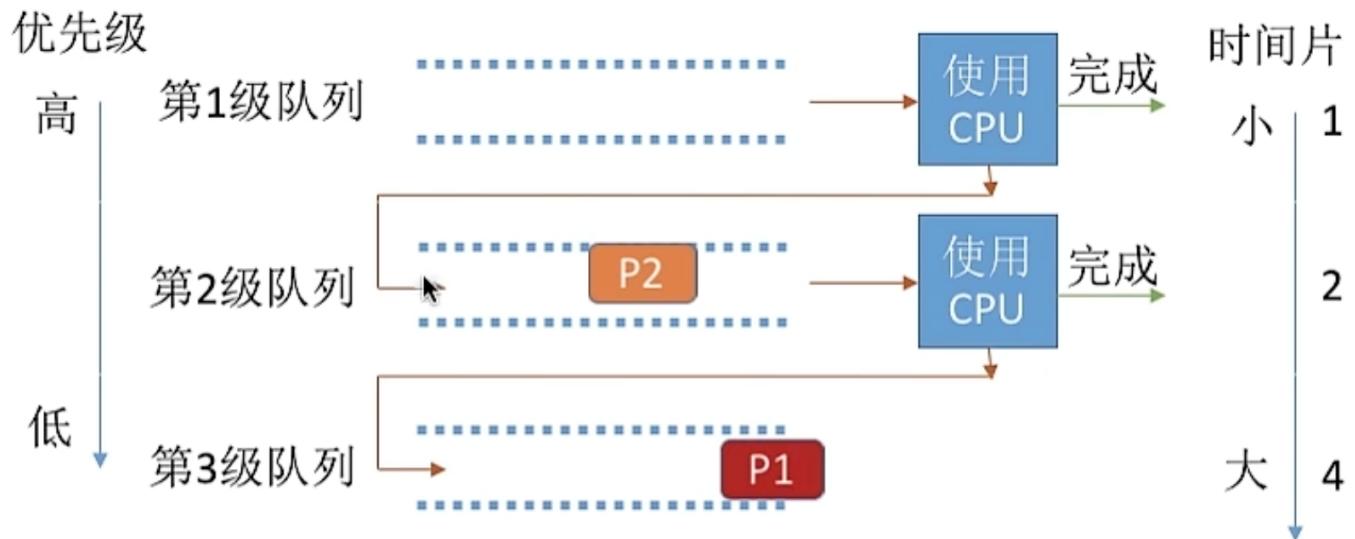
多级反馈队列调度

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。
优点	队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

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

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



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

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

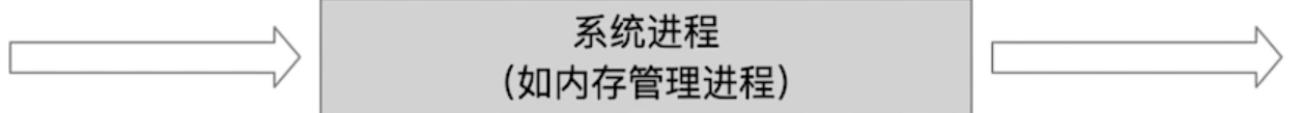
算法	思想&规则	可抢占？	优点	缺点	会导致饥饿？	补充
时间片轮转		抢占式	公平，适用于分时系统	频繁切换有开销，不区分优先级	不会	时间片太大或太小有何影响？
优先级调度		有抢占式的，也有非抢占式的。注意做题时的区别	区分优先级，适用于实时系统	可能导致饥饿	会	动态/静态优先级。各类型进程如何设置优先级？如何调整优先级？
多级反馈队列	较复杂，注意理解	抢占式	平衡优秀 666	一般不说它有缺点，不过可能导致饥饿	会	

注：比去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）

多级队列调度

系统中按进程类型设置多个队列，进程创建成功后插入某个队列

最高优先级



交互式进程
(如游戏、打字软件)

批处理进程
(如AI模型训练、视频特效渲染)

最低优先级

多级队列调度

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%

各队列可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