考研 408—操作系统讲义【处理机调度】

一. 处理机调度

1. 原因：在多道程序环境下，内存中的进程数目往往多于处理机数目。
2. 目的：在多道程序环境下，动态的将 cpu 按照某种算法分配给就绪状态的进程。
3. 处理机调度的层次：

I. 高级调度（作业调度，长程调度）
   i) 适用于批处理系统（大多数）
   ii) 作用：将用户提交的存放于磁盘后备队列中的作业调入内存。
   ⟹分配必要的资源⟹创建进程
   iii) 缺点：运行频率较低（n 分钟一次）
   ➢ **分时，实施系统不用。**

II. 低级调度（进程调度，短程调度）
   i) 作用：决定就绪队列中哪个进程优先获得处理机并将处理机资源分配给该进程
   ii) 是最基本的调度，三种类型的操作系统都必须配备。
   运行频率高（几十毫秒）所以算法不可太复杂。
   iii) 主要任务
   A. 保护当前进程的处理机现场。
   B. 按照调度算法选取投入执行的新进程。
   C. 将处理机分配给新进程。
   iv) 进程调度的两种方式
   A. 非抢占方式
   ——除非进程完成或发生某事件而阻塞，否则进程一直占据 CPU。
   ——优点：简单，系统开销小。
   ——缺点：难以满足紧急任务，不适宜要求较严格的操作系统。
   B. 抢占方式
   ——优先权原则
   ——短作业优先原则
   ——时间片原则

III. 中级调度（内存调度，中程调度）
   i) 作用：
   A. 将外存中具备运行条件的进程换入内存
   B. 将内存中处于阻塞状态的进程换入外存
   ii) 目的：解决内存紧张问题
   iii) 常用于分时系统及其有虚拟存储器的系统中，运行频率介于高低之间。

二. 处理机调度目标

调度方式和算法的选择取决于操作系统的类型及其目标，一般系统都要求调度能达到系统吞吐量大、处理机利用率高、各类资源能平衡利用以及对不同类型的作业具有公平性等目标。

但是不同类型的系统通常还会有自己的要求，譬如，批处理系统希望周转时间短，分时系统要求响应时间快，而实时系统则要求能保证其截止时间等，因此它们将采用不同的调度方式和调度算法。

三. 进程调度及调度算法中的典型问题分析

【例 1】引起进程调度的因素有哪些?

答:引起进程调度的因素有:

(1) 正在执行的进程正常终止或异常终止。

(2) 正在执行的进程因某种原因而阻塞。具体包括：

- 提出 I/O 请求后被阻塞;
- 在调用 wait 操作时因资源不足而阻塞;
- 因其他原因执行 block 原语而阻塞等。

(3) 在引入时间片的系统中，时间片用完。

(4) 在抢占调度方式中，就绪队列中某进程的优先权变得比当前正在执行的进程高，或者有优先权更高的进程进入就绪队列。

四. 调度算法

1. 先来先服务算法（FCFS）
2. 短作业优先（SJF）
3. 高响应比优先调度算法（HRRN）
4. 时间片轮转算法（RR）
5. 多级队列调度算法
6. 多级反馈队列调度算法

实时调度算法

I. 最早截止时间优先（EDF）

II. 最低松弛度优先（CCF）

1. 先来先服务算法

FCFS 是最简单的调度算法，既可以用作作业调度，也可以用作进程调度

这种算法优先考虑系统中等待时间最长的作业（进程），而不管作业所需执行时间长短，

做法是从后备队列中选择几个最先进入该队列的作业，将它们调入内存，为它们分配资源和创建进程，然后放入就绪队列

进程调度中使用此算法时，每次都从就绪的进程队列中选择一个最先进入该队列的进程，为之分配处理机，使之投入运行，该进程会一直运行到完成或者因发生某事件而阻塞后，进程调度程序才会把处理机分配给其他进程

2. 短作业优先（SJF）

由于在实际情况中短作业（进程）所占比例很大，为了让它们比长作业优先执行，就有了此算法。

SJF 顾名思义以作业长短来确定优先级（作业越短优先级越高），作业的长短用作业所需的运行时间来衡量，此算法一样也可以用做进程调度，它将从外存的作业后备队列中选择若干个估计运行时间最短的作业，优先将它们调入内存运行。

SJF 调度算法也存在不容忽视的缺点：该算法对长作业不利，如作业 C 的周转时间由 10 增至16，其带权周转时间由 2 增至3.1。更严重的是，如果有一长作业(进程)进入系统的后备队列(就绪队列)，由于调度程序总是优先调度那些(即使是后进来的)短作业(进程)，将导致长作业(进程)长期不被调度。该算法完全未考虑作业的紧迫程度，因而不能保证紧迫性作业(进程)会被及时处理。由于作业(进程)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作业优先调度。

3. HRRN（高响应比优先算法）

I. 算法思想：要综合考虑作业/进程的等待时间和要求服务时间

II. 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

III. 响应比：响应比=（等待时间+要求服务时间）/要求服务时间

IV. 用于调度：即可以用于作业调度，也可以用于进程调度

V. 是否可以抢占：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。

VI. 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。

4. 轮转调度算法 Round-Robin(RR)

在分时系统中都采用时间片轮转算法进行进程调度。时间片是指一个较小的时间间隔，通常为 10-100 毫秒。在简单的轮转算法中，系统将所有的就绪进程按先来先服务（即 FIFO）规则排成一个队列，将 CPU 分配给队首进程，且规定每个进程最多允许运行一个时间片；若时间片使用完进程还没有结束，则被加入就绪 FIFO 队列队尾，并把 CPU 交给下一个进程。时间片轮转算法只用于进程调度，它属于抢占调度方式。

优点：CPU 分配相对公平；平均响应时间较短

缺点：不利于紧急作业，而且当进程的运行时间都相近时，平均的等待时间比较长，甚至不如先来先服务算法。

5. 多级反馈队列算法

多级反馈队列算法：不需要事先知道各种进程所需要的执行时间，还可以较好地满足各种类型进程的需要，是目前公认的一种较好的进程调度算法。

I. 调度机制：

i) 设置多个就绪队列。在系统中设置多个就绪队列，并为每个队列赋予不同的优先级，从第一个开始逐个降低。不同队列进程中所赋予的执行时间也不同，优先级越高，时间片越小。

ii) 每个队列都采用 FCFS（先来先服务）算法。轮到该进程执行时，若在该时间片内

完成，便撤离操作系统，否则调度程序将其转入第二队列的末尾等待调度，……。若进程最后被调到第 N 队列中时，便采用 RR 方式运行。

iii) 按队列优先级调度。调度按照优先级最高队列中诸进程运行，仅当第一队列空闲时才调度第二队列进程执行。若优先级低队列执行中有优先级高队列进程执行，应立刻将此进程放入队列末尾，把处理机分配给新到高优先级进程。

6. 实时调度算法按调度方式分为：

I. 非抢占式算法：

i) 非抢占式轮转调度算法。

ii) 非抢占式优先调度算法。

II. 抢占式调度算：

i) 基于时钟中断的抢占式优先级调度算法。

ii) 立即抢占的优先级调度算法

实时调度算法

最早截止时间优先（EDF）

最早截止时间优先 EDF（Earliest DeadlineFirst）算法是非常著名的实时调度算法之一。在每一个新的就绪状态，调度器都是从那些已就绪但还没有完全处理完毕的任务中选择最早截止时间的任务，并将执行该任务所需的资源分配给它。在有新任务到来时，调度器必须立即计算 EDF，排出新的定序，即正在运行的任务被剥夺，并且按照新任务的截止时间决定是否调度该新任务。如果新任务的最后期限早于被中断的当前任务，就立即处理新任务。按照 EDF 算法，被中断任务的处理将在稍后继续进行。

2. 该算法的思想是从两个任务中选择截至时间最早的任务,把它暂作为当前处理任务,再判断该任务是否在当前周期内,若不在当前周期内,就让另一任务暂作当前处理任务,若该任务也不在当前周期内,就让 CPU 空跑到最靠近的下一个截至时间的开始,若有任务在该周期内,就判断该任务的剩余时间是否小于当前截至时间与当前时间的差,若小于,则让该任务运行到结束.否则,就让该任务运行到该周期的截止时间,就立即抢回处理器,再判断紧接着的最早截至时间,并把处理器给它,做法同上,如此反复执行.

最低松弛度优先（LLF）

最低松弛度优先（LLF）算法是根据任务紧急（或松弛）的程度，来确定任务的优先级。任务的紧急程度愈高，为该任务所赋予的优先级就愈高，使之优先执行。在实现该算法时要求系统中有一个按松弛度排序的实时任务就绪队列，松弛度最低的任务排在队列最前面，被优先调度。松弛度的计算方法如下：

任务的松弛度=必须完成的时间-其本身的运行时间-当前时间

其中其本身运行的时间指任务运行结束还需多少时间，如果任务已经运行了一部分，则：任务松弛度=任务的处理时间-任务已经运行的时间–当前时间

几个注意点：

1. 该算法主要用于可抢占调度方式中，当一任务的最低松弛度减为 0 时，它必须立即抢占 CPU，以保证按截止时间的要求完成任务。

2. 计算关键时间点的各进程周期的松弛度，当进程在当前周期截止时间前完成了任务，则在该进程进入下个周期前，无需计算它的松弛度。

3. 当出现多个进程松弛度相同且为最小时，按照“最近最久未调度”的原则进行进程调度。

例题：

假设一个系统中有 5 个进程,它们的到达时间和服务时间如图所示，忽略 I/O 以及其他开销时间,若分别按先来先服务(FCFS)、非抢占及抢占的短作业优先(SJF)、高响应比优先(HRRN)、时同片轮转(RR,时间片=1)、多级反馈队列调度算法(FB,第 i 级队列的时间片=2^{i-1})以及立即抢占的多级反馈队列调度算法(FB,第 i 级队列的时间片=2^{i-1})进行 CPU 调度,请给出各进程的完成时间、周转时间、带权周转时间、平均周转时间和平均带权周转时间。

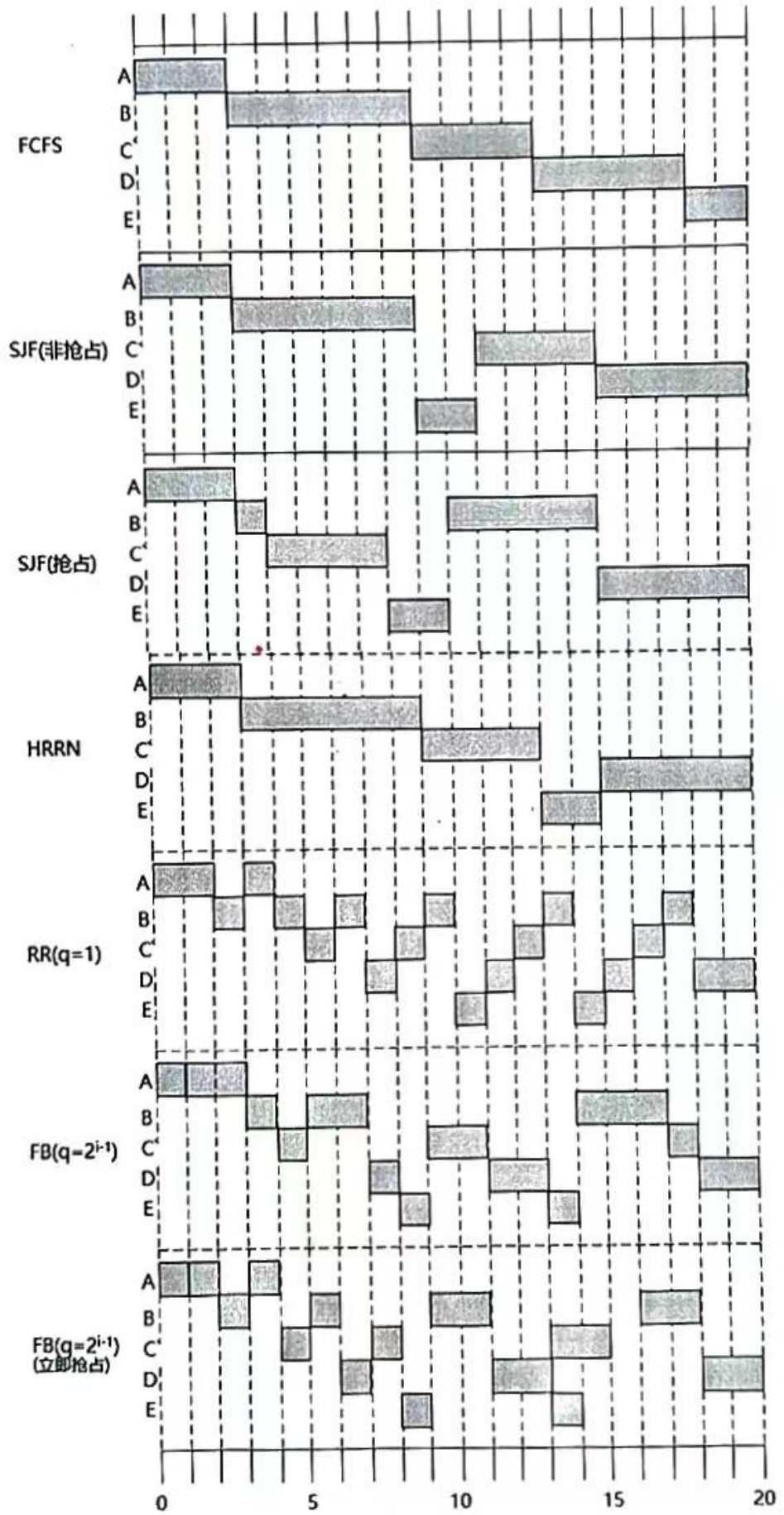
进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

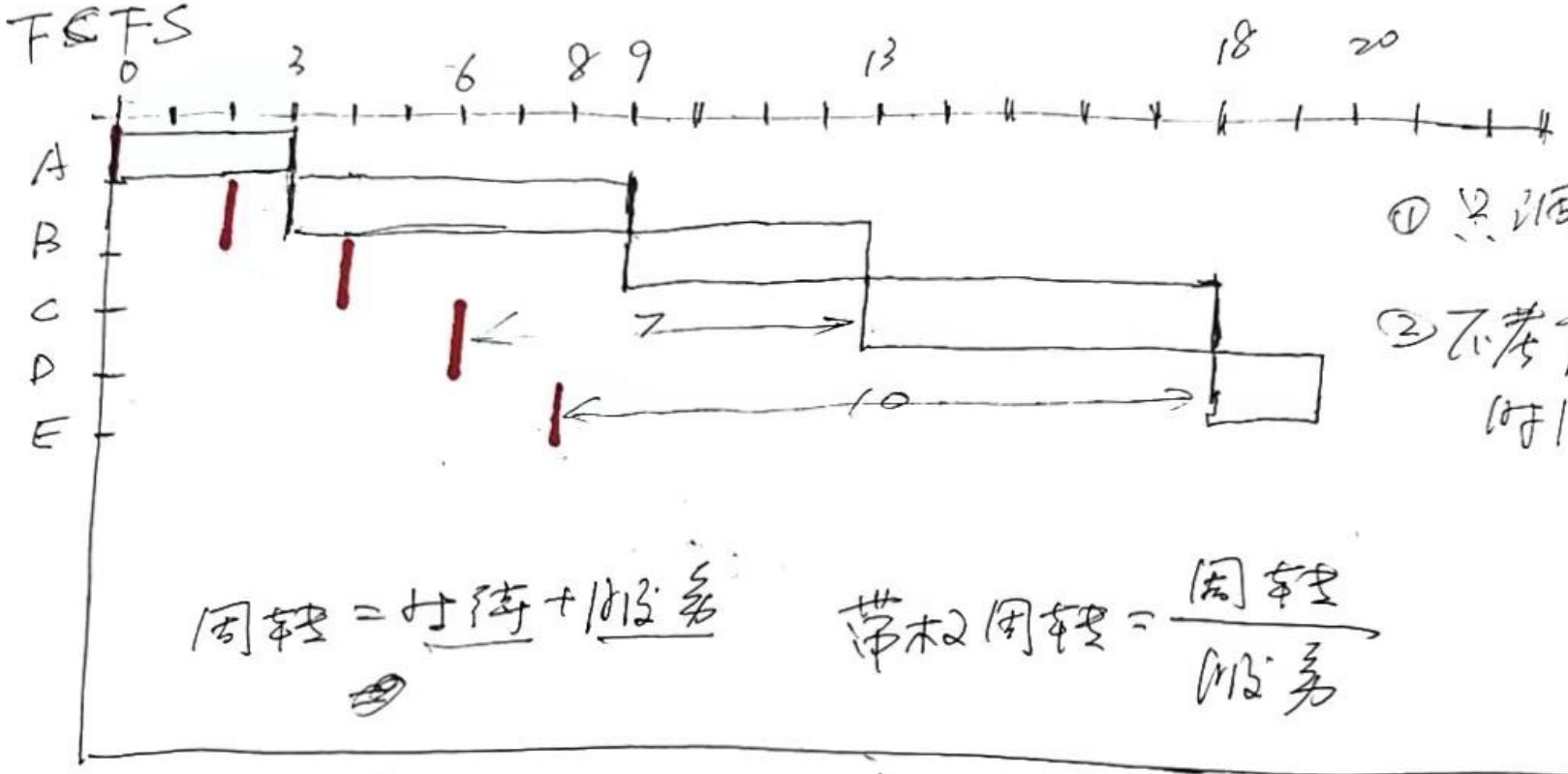
分析:

进程调度的关键是理解和掌握调度所采用的算法，FCFS 算法选择最早进入就绪队列的进程投入执行;SJF 算法选择估计运行时间最短的进程投入执行,采用抢占方式时,若新就绪的进程运行时间比正在执行的进程的剩余运行时间短,则新进程将抢占 CPU;HRRN 算法选择响应比（响应比=（运行时间＋等待时间）/运行时间）最高的进程投入执行;RR 算法中，就绪进程按 FIFO 方式排队,CPU 总是分配给队首的进程，并只能执行一个时间片;FB 算法将就绪进程排成多个不同优先权及时间片的队列，新就绪进程总是按 FIFO 方式先进入优先权最高的队列，CPU 也总是分配给较高优先权队列上的队首进程,若执行一个时间片仍未完成，则转入下一级队列的末尾，最后一级队列则采用时间片轮转方式进行调度。

答:

对上述 5 个进程按各种调度算法调度的结果如图所示，从中可以计算出各进程的完成时间、周转时间和平均周转时间(如表所示)。


FCFS
SJF(非抢占)
SJF(抢占)
HRRN
RR(q=1)
FB(q=2^(i-1))
FB(q=2^(i-1))
(立即抢占)
A
B
C
D
E
0
5
10
15
20



周转 = 时待 + 服务

带权周转 = 周转 / 服务

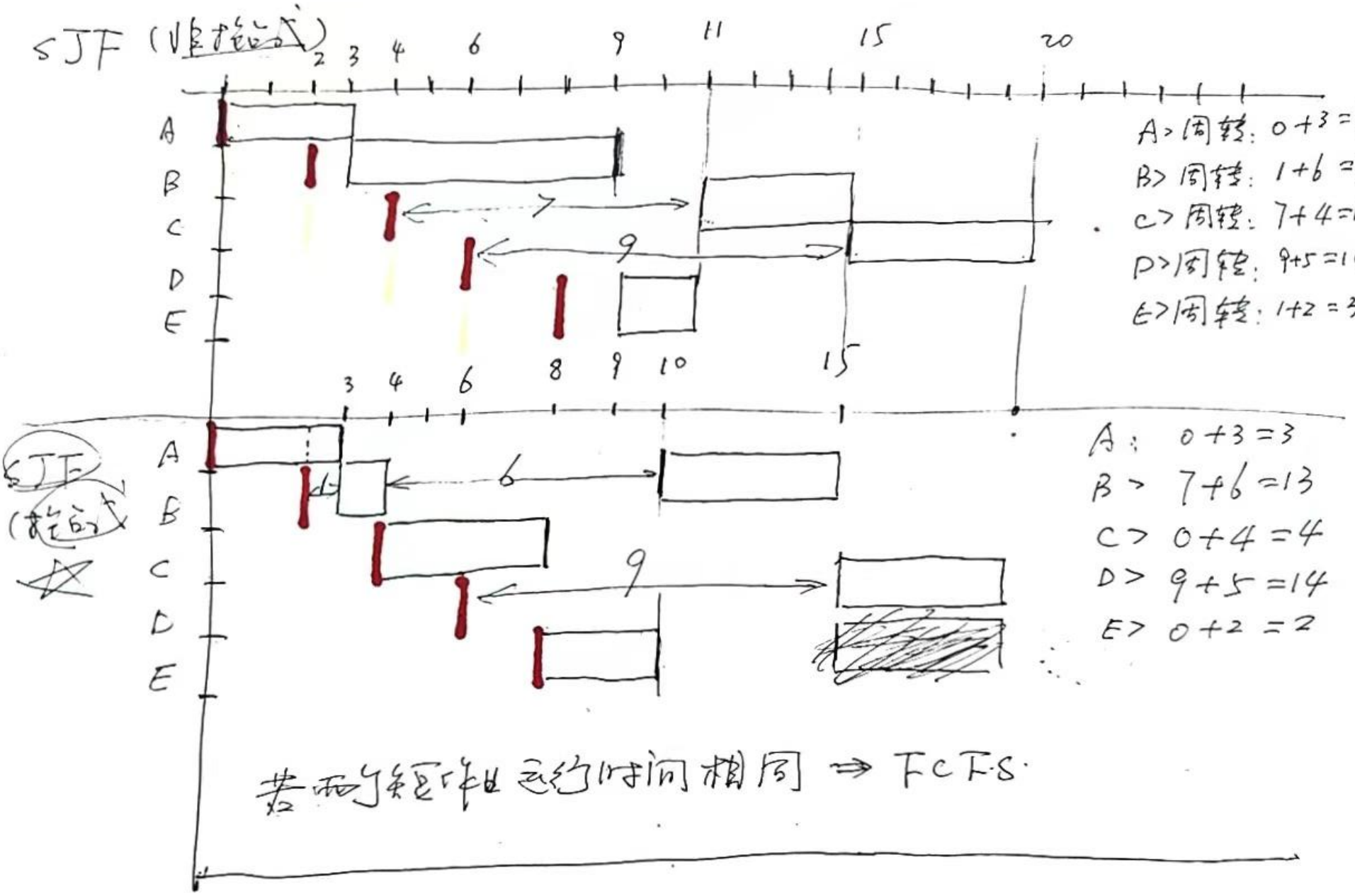
A
- 时待：0
- 周转 3
- 带权周转：$\frac{3}{3}=1$

B
- 时：1
- 周转：7
- 带权周转：$\frac{7}{6}=1.17$

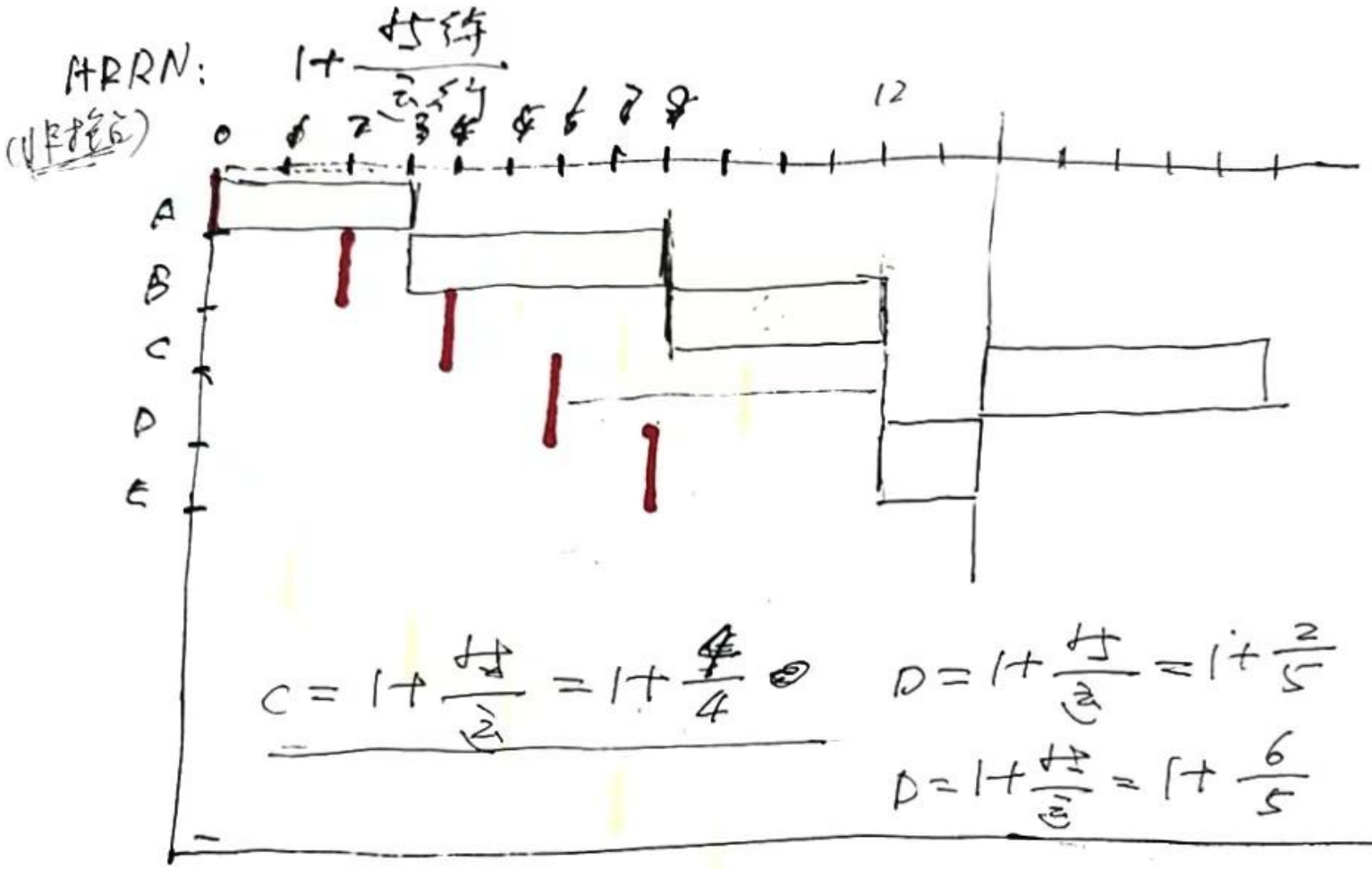
C
- 时：5
- 周转：5+4=9
- 带权：$\frac{9}{4}=2.25$

D
- 时：7
- 周转：7+5=12
- 带权周转：$\frac{12}{5}=2.4$

E
- 时：10
- 周：12
- 带权周转：$\frac{12}{2}=6$


SJF (非抢占式)
2 3 4 6 9 11 15 20
A
B
C
D
E
9
A> 周转: 0+3=3
B> 周转: 1+6=7
C> 周转: 7+4=11
D> 周转: 9+5=14
E> 周转: 1+2=3
3 4 6 8 9 10 15
SJF
(抢占式)
6
9
A: 0+3=3
B> 7+6=13
C> 0+4=4
D> 9+5=14
E> 0+2=2
若所有作业运行时间相同 ⇒ FCFS.

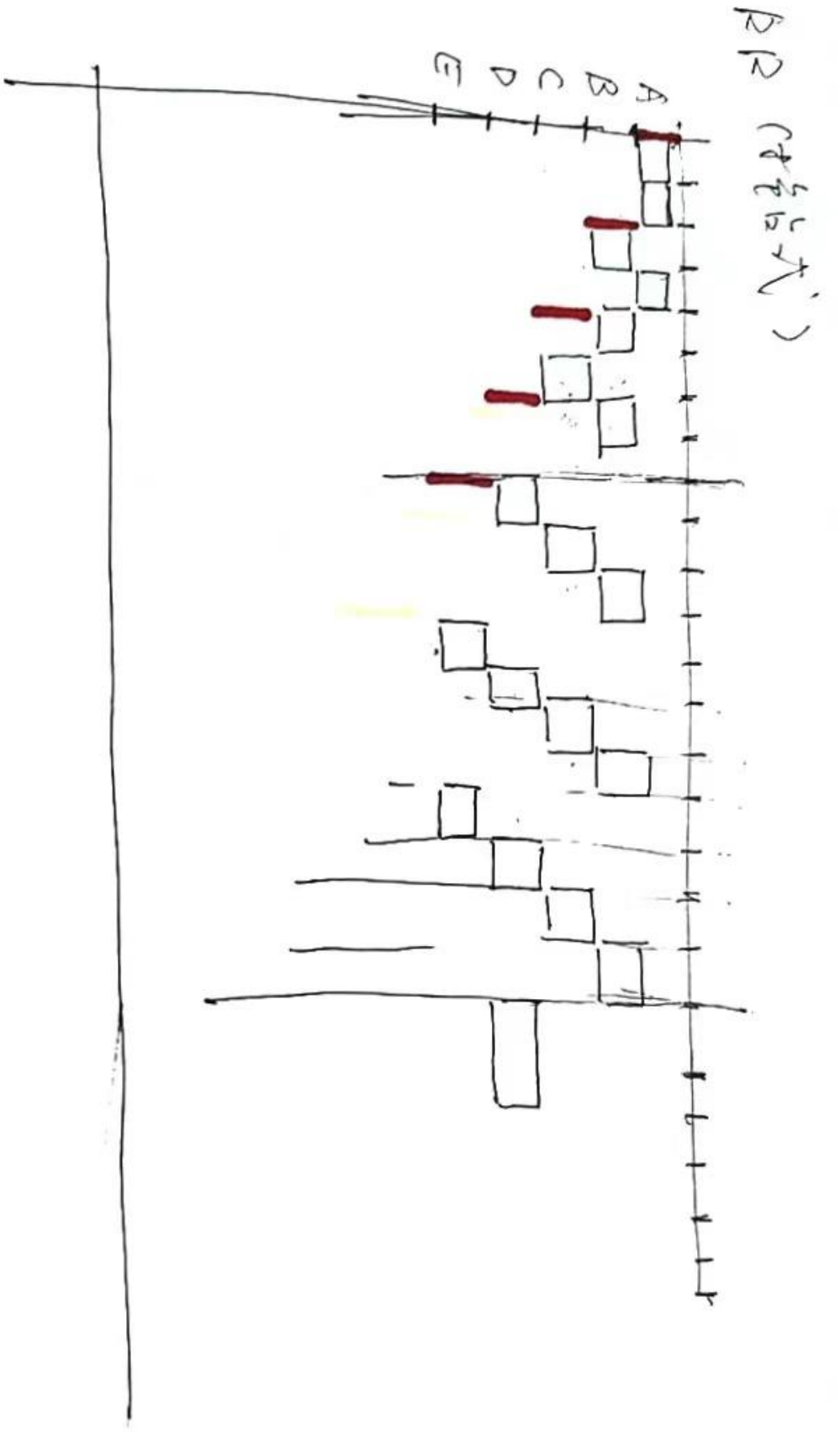
HRRN: $1+\frac{\text{等待}}{\text{运行}}$



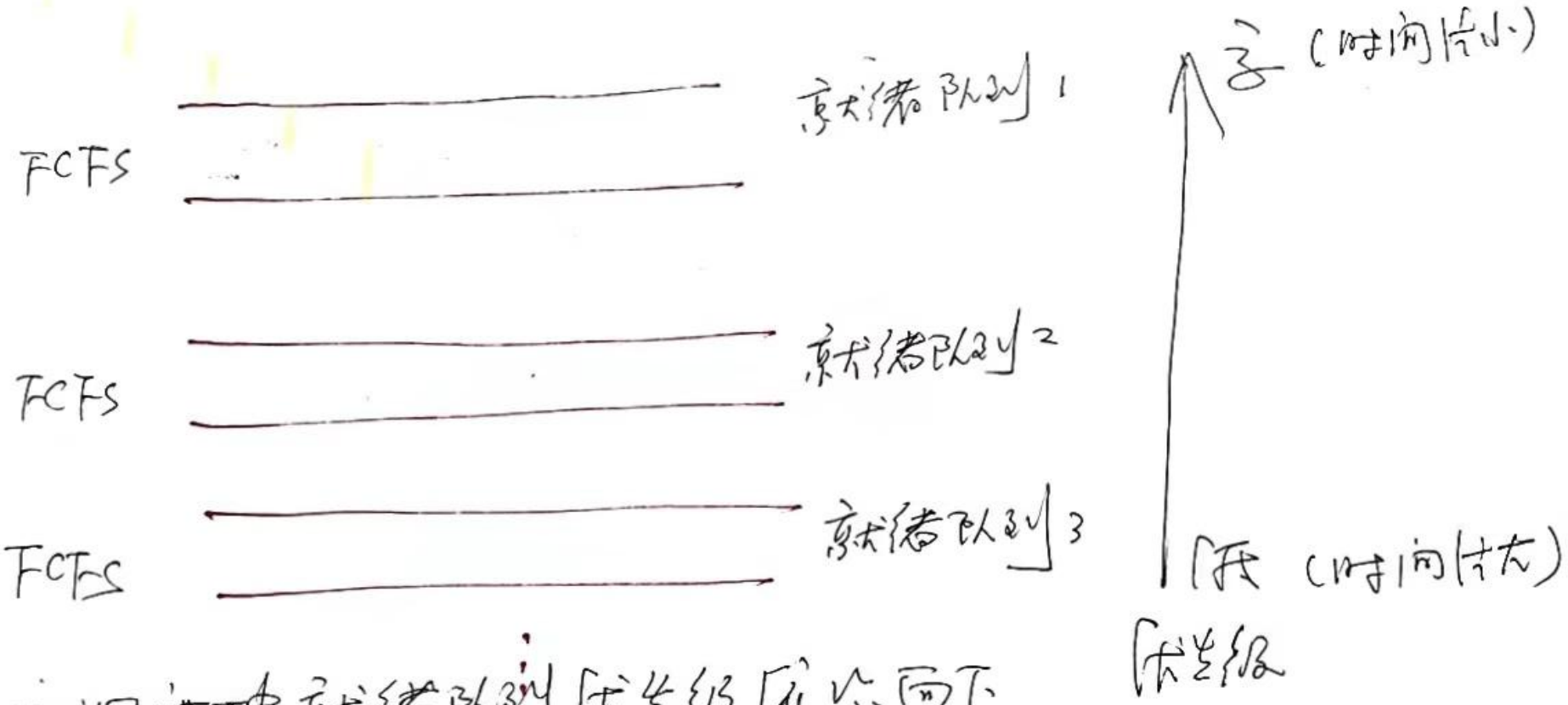
$C=1+\frac{4}{2}=1+\frac{4}{4}=2$

$D=1+\frac{4}{2}=1+\frac{2}{5}$　　$E=1+\frac{2}{2}=1+\frac{0}{2}$

$D=1+\frac{4}{2}=1+\frac{6}{5}$　　$E=1+\frac{4}{2}=1+\frac{4}{2}$


A
B
C
D
E

多级反馈队列算法



① 调度时各就绪队列优先级依次向下

② 新进程调度后若未完，则进入下一级就绪队列

③ 各个就绪队列内为FCFS调度

A
B
C
D
E
I.
II.
III.
5ms
6ms
5ms

2018 — 24

	计算	等待
P1	30ms	12ms
P2	15ms	24ms
P3	18ms	36ms

~~计算 等待~~

~~P1 计算 30 12~~

~~P2~~

~~P3 30+18+1+1+1~~

平均周转时间

$$= \frac{P1+P2+P3}{3}$$

$$= \frac{80+36+105}{3} =$$

计算 等待

P1 $30ms+1ms+24ms+1ms+36ms+1ms+12ms$ $105ms$

P2 $15+1ms$ $24 = 40ms$

P3 ~~15+1ms~~ $18ms+1ms+24ms+1ms+36ms$

$= 80ms$

2019 — 27

P1 30ms

P2 20ms

① Q1 (时间片轮转) (10ms)

P1 P2

Q2 (SJF).

P1 P2

P1时 = 20ms.

P2时 = 10ms

平均时待时间

$= \frac{P1时 + P2时}{2} = \frac{20+10}{2}$

$= 15ms.$

⇒ P1: 时=0

运=10ms.

P2: 时=10ms

运=10ms

P2作业: 时待 (10ms) +10ms

运行 20ms.

P2作业: 时待 0

运行 (10ms) 10ms.

①
多个线程 — 1个核.
(2)
P1、P2 轮流
P1
P2
申请
释放
打印机
读卡机
P1、P2 都在
P1
P2
申请
释放
打印机
读卡机
会
发生
死锁
吗?
会发生,
一旦进程P1占有打印机,进程P2占有读卡机,
P1和P2都会因为申请不到对方的资源而
进入阻塞状态.

死锁产生原因：

① 竞争资源 —— 共享资源不足

—— 资源分配不当使用

② 进程推进顺序非法 —— 银行家算法

产生死锁的必要条件

① 互斥条件 —— 所分资源不能使用

② 请求与保持条件

③ 不剥夺条件

④ 环路等待 ☆ —— 每个进程都阻塞状态

（进程之间）所分进程之间进入环路等待 ②

必要条

☆判断死锁依据

（进程集合中若只有部分进程陷入阻塞状态，不可判断为死锁）

☆ 预防死锁 —— 破坏产生死锁的4个必要条件.

① 互斥条件 → 进程之间要并发执行的必要条件.
—— 不允许破坏. 必须要的.
☆ 类似于打印机. 可以通过spooling技术破坏.

② 占有且等待
→ 系统要求每个进程一次性申请所有的资源
优点: 不会死锁
缺点: ① 资源浪费 ⇒ 利用率低
② 有些进程因为申请不到资源而无法推进. —— 饥饿现象.

③ 非抢占
→ 进程已经占有的资源, 未使用完之前, 不能被抢占, 只能在使用完后由自己释放.
优点: 资源利用率高.
缺点: ① 实现复杂 ⇒ 代价大
② 进程推进前面的资源可能反复释放和申请, 会增大系统开销.

③ 可屏蔽
——：当一个进程中断处理程序执行了一段时间
后可以屏蔽掉
必须释放所占资源

在是：1> 多道程序为，
2> 七个如多道程序引，缺的
——P中「所多道程序」事

☆④ 摆脱环路等待

方> 指系统类型为每个资源分配不同的序号

每个进程按资源序号的递增顺序申请资源

——给资源分配序号

方——：系统最好的利用率.

☆ 银行家算法 —— 避免死锁

每次分配资源时先进行安全性检查 ⇒ 进程序列

⇒ 按进程序列 调度进程 分配资源 ☆

(银行家算法只确定一个进程序列，并不真正分配资源)

(只有确定进程序列以后，再由调度算法调度进程时才按照银行家算法的结果分配资源)

(银行家算法不一定可以找到一个进程序列)

数据结构：

Available —— 系统中现有
Max —— 表示 n 个进程，m 个资源的最大需求量
Allocation —— 每个进程已分配的资源数
Need —— 每个进程还需要的资源数

Need[i,j] = Max[i,j] - Allocation[i,j]

安全性算法：

Work：安全性算法执行时，系统可利用资源数目
Finish：表示是否有足够资源分配给进程
初始：false

例：在银行家算法中，若有下面的资源分配：

进程	MAX A B C D	Need A B C D	Available A B C D
P1	0 0 4 4	0 0 1 2	1 6 2 2
P2	2 7 5 0	1 7 5 0	
P3	3 6 10 10	2 3 5 6	
P4	0 9 8 4	0 6 5 2	
P5	0 6 6 10	0 6 5 6	

① 为什么从 P1 开始？

① 是否存在一个安全序列？ 现在 P1→P4→P5→P2→P3

进程	work	Need（尚需）	Allocation	work+Allocation	Finish
√P1	1 6 2 2	0 0 1 2	0 0 3 2	1 6 5 4	T
P4 P2	1 6 5 4	0 6 5 2	0 3 3 2	1 9 8 6	T
P5 P3	1 9 8 6	0 6 5 6	0 0 1 4	1 9 9 10	T
P2 P4	1 9 9 10	1 7 5 0	1 0 0 0	2 9 9 10	T
P3 P5	2 9 9 10	2 3 5 6	1 3 5 4	3 12 14 14	T

☆ 可以存在多个安全序列

☆ 也可以找不到一个安全序列

② 若进程 P3 发出 request (1, 2, 2, 2)，系统可否为它分配资源？ ⇒ 在需求范围内表示请求合理

① 检查 request 是否在总需求范围内

request (1, 2, 2, 2) ≤ Need$_{P3}$ (2, 3, 5, 6)

② 系统是否可为 P3 分配？

request$_{P3}$ (1, 2, 2, 2) ≤ Available (1, 6, 2, 2)

— 可以给

③ 安全性检查：[假设]将 Request P3(1,2,2,2) 分配给 P3.

Available = (0,4,0,0)

P3: Allocation = (2,5,7,6)

P3: Need = (1,1,3,4)

结论：系统无法对进程请求 Request (1,2,2,2) 分配给 P3

④ 若系统对进程请求，系统是否会进入死锁？ 不会

① 进程再次申请资源管理

② P3 进入阻塞状态

☆ 死锁：多个进程都得不到管理而陷入的无限时等待状态