



经典教材《计算机操作系统》**最新版**

# 第3章 处理机调度与死锁

主讲教师：王申

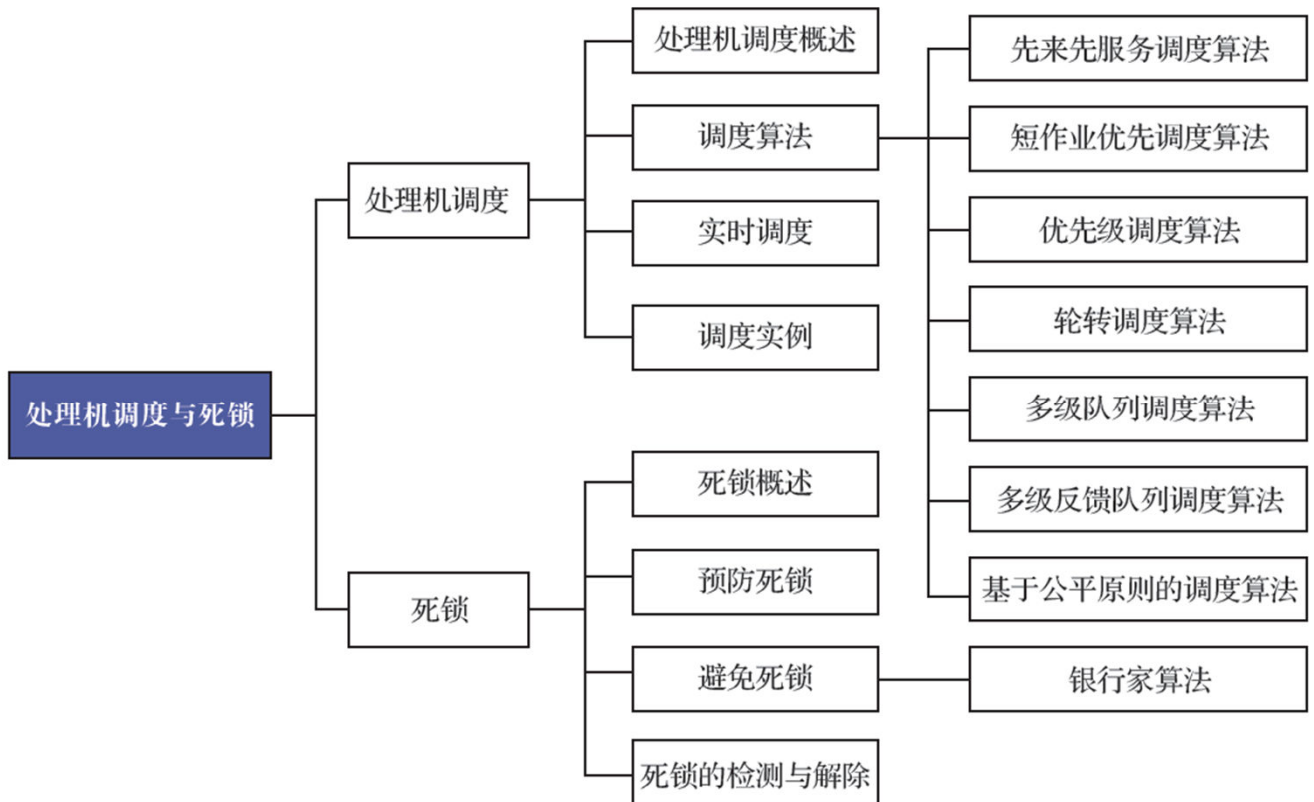


 **人民邮电出版社**  
POSTS & TELECOM PRESS











## 第3章知识导图

第1章	操作系统引论
第2章	进程的描述与控制
第3章	处理机调度与死锁
第4章	进程同步
第5章	存储器管理
第6章	虚拟存储器
第7章	输入/输出系统
第8章	文件管理
第9章	磁盘存储器管理
第10章	多处理机操作系统
第11章	虚拟化和云计算
第12章	保护和安全





## 内容导航:

-  **3.1 处理机调度概述**
-  3.2 调度算法
-  3.3 实时调度
-  3.4 Linux进程调度
-  3.5 死锁概述
-  3.6 预防死锁
-  3.7 避免死锁
-  3.8 死锁的检测与解除

# 第3章 处理机调度与死锁

---

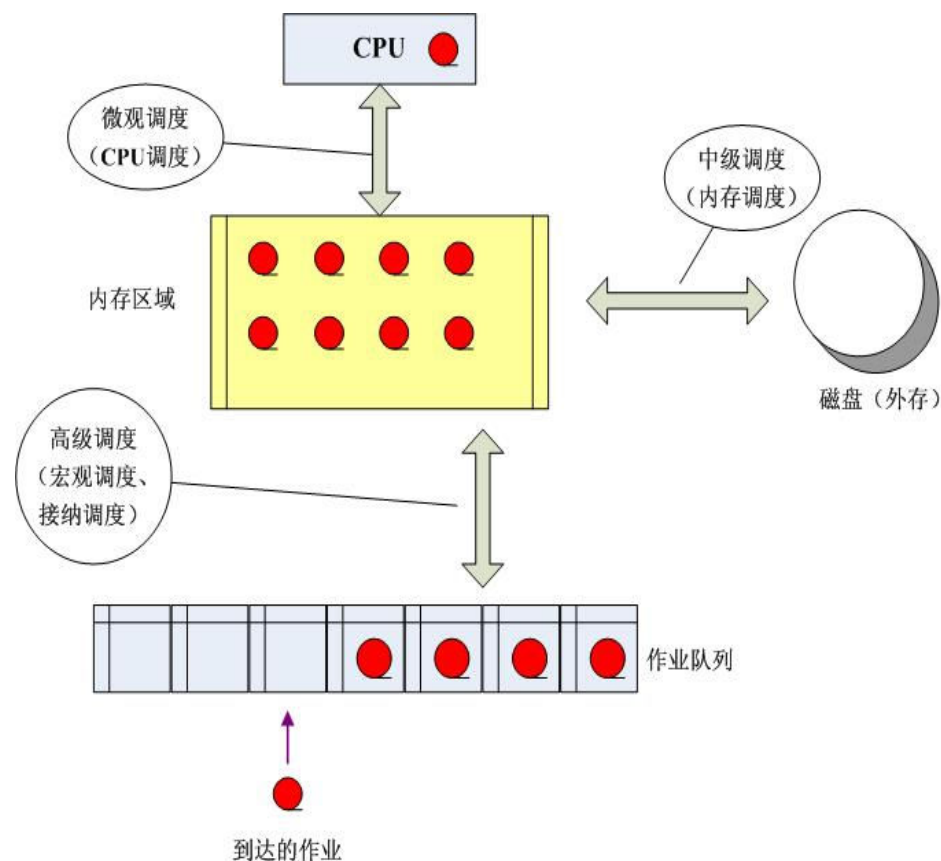
1. 在多道程序系统中，一个作业从提交到执行，通常

都要经历**多级调度**

- 如高级调度、低级调度、中级调度以及I / O调度等

2. 系统的**运行性能**在很大程度上取决于调度

- 如吞吐量的大小、周转时间的长短、响应的及时性等



## 3. 调度是多道程序系统的关键

### ➤ CPU资源管理——多道程序设计面临的挑战

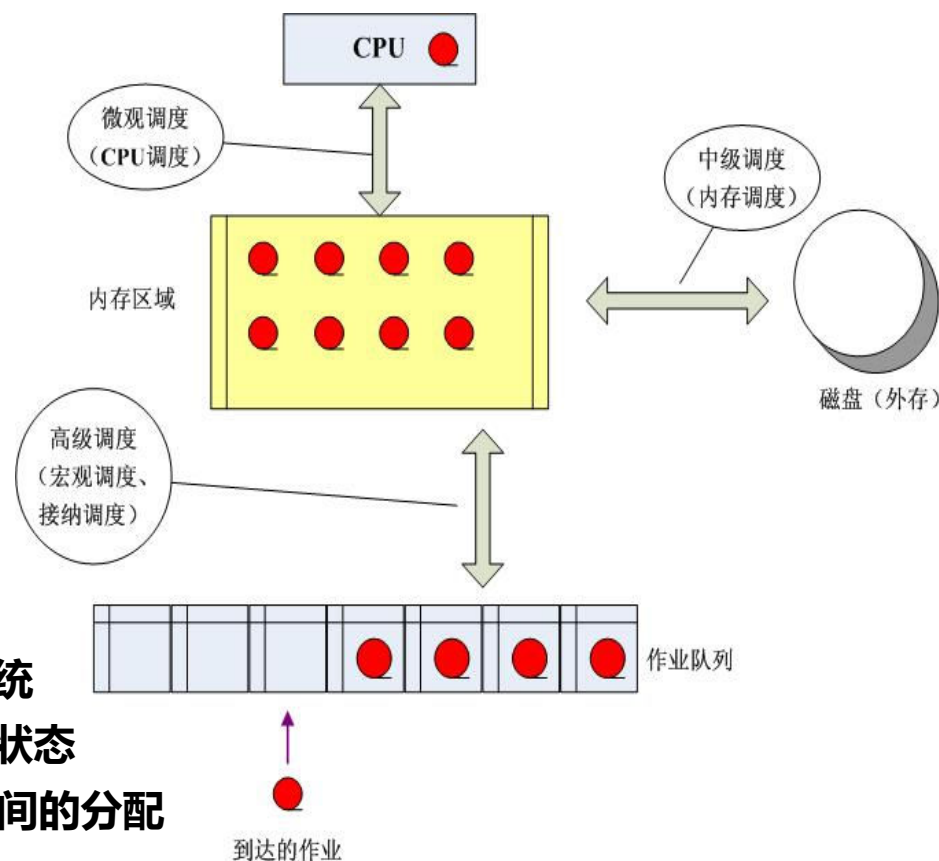
- 批处理系统：如何安排内存中多个作业的运行顺序？
- 交互式系统：如何更好应对不同的交互式请求？
- 实时系统：如何保证实时服务的高质量？

### ➤ 进程调度——有效的管理CPU资源

- When：何时进行进程调度？
- How：遵循何种规则完成调度？
- What：调度过程中需要完成哪些工作？

### ➤ 进程调度的级别

- 高级调度：也称宏观调度，决定哪些程序可以进入系统
- 中级调度：也称内存调度，决定内存中程序的位置和状态
- 低级调度：也称微观调度，决定CPU资源在就绪进程间的分配





## 处理机调度层次



高级调度（长程调度/作业调度）



低级调度（短程调度/进程调度）



中级调度（中程调度/内存调度）



## 高级调度

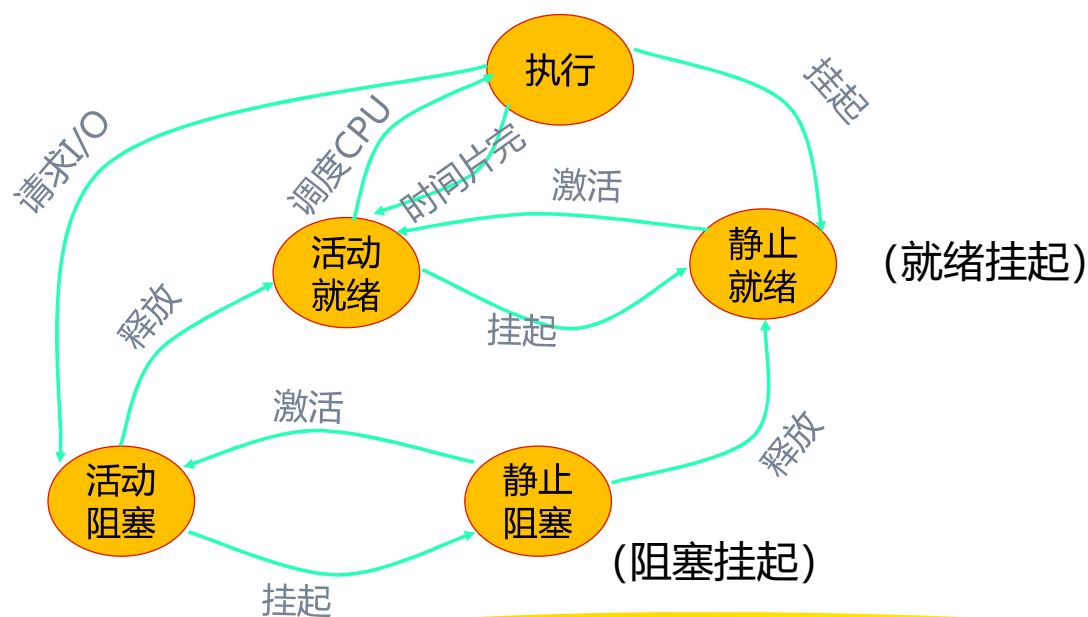
调度对象：作业

根据某种算法，决定将外存上处于后备队列中的作业调入内存，并为它们创建进程和分配必要的资源。然后，将新创建的进程排在就绪队列上等待调度。

主要用于多道批处理系统中，分时和实时系统不设置高级调度。



引入中级调度的**主要目的**，是**为了提高内存利用率和系统吞吐量**；  
使那些**暂时不能运行**的进程不再占用宝贵的内存资源，将它们调至外存上去等待，把此时的进程状态称为**就绪驻外存状态或挂起状态**；







引入中级调度的**主要目的**，**是为了提高内存利用率和系统吞吐量**；  
使那些**暂时不能运行**的进程不再占用宝贵的内存资源，将它们调至外存上去等待，把此时的进程状态称为**就绪驻外存状态或挂起状态**；



当这些进程具备运行条件、且内存又稍有空闲时，由**中级调度来决定把就绪进程，重新调入内存，并修改其状态为就绪状态**，挂在就绪队列上等待进程调度；



即“**对换**”功能；短期**调整系统负荷**，平顺系统操作

将在第5章 存储器管理中介绍



## 低级调度



调度对象：进程



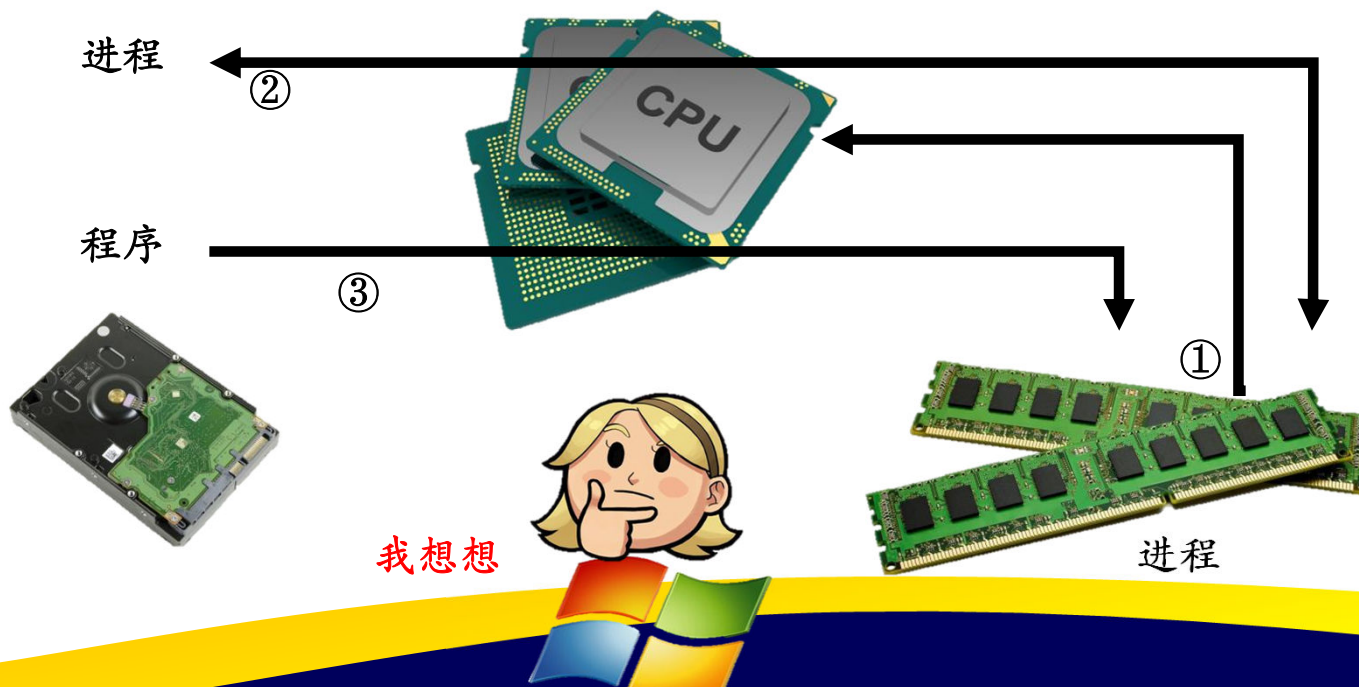
根据某种调度算法，决定就绪队列中的哪个进程应获得处理机



应用在于多道批处理、分时和实时OS

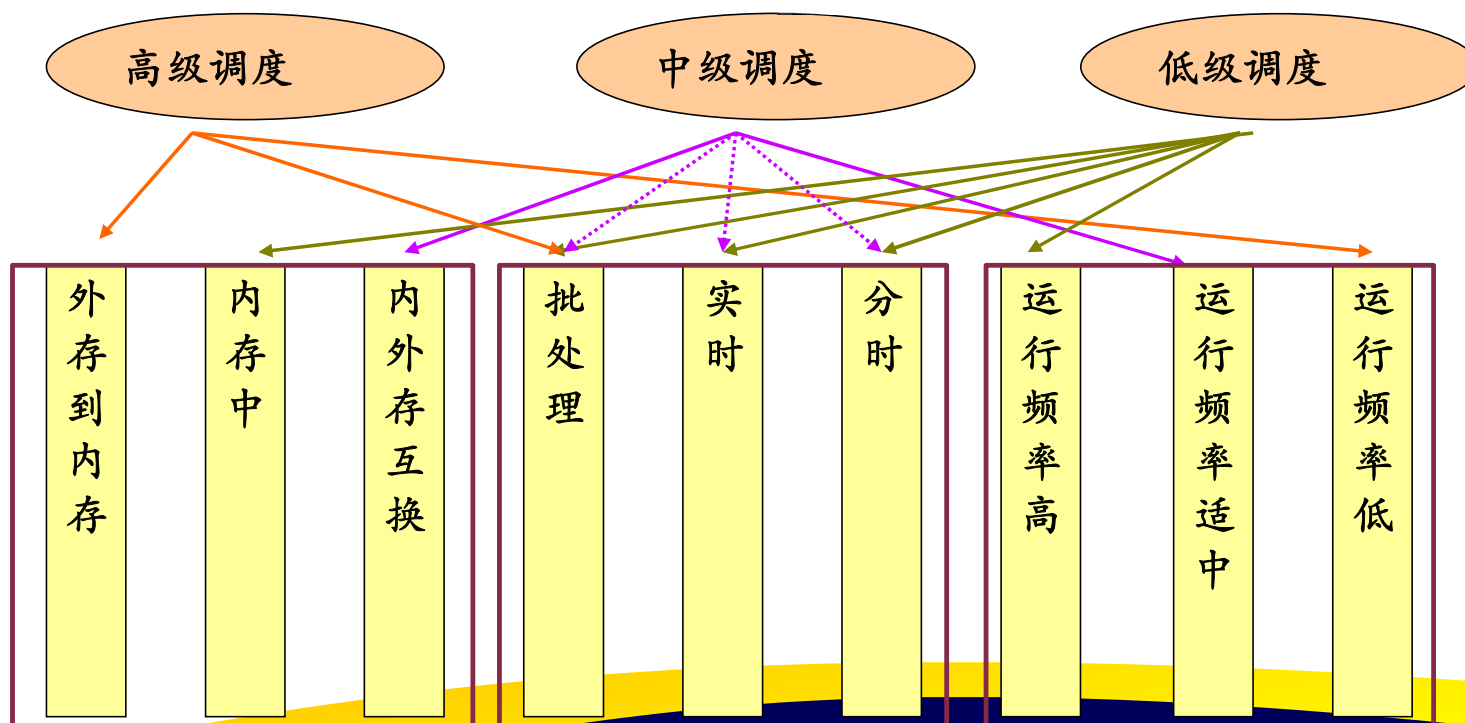
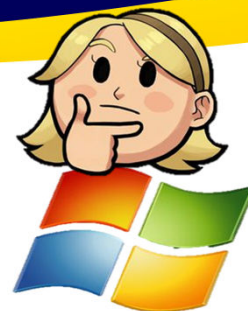
# 处理机调度

调度类型	运行频率	运行时间	算法复杂性
①低级调度 (占用CPU)	高 (毫秒级)	短	低 (不能占用太多cpu时间)
②中级调度 (内存 $\leftrightarrow$ 外存)	中等	较短	中等
③高级调度 (外存 $\rightarrow$ 内存)	低 (分钟级)	长	高 (允许复杂的调度算法)



# 处理机调度

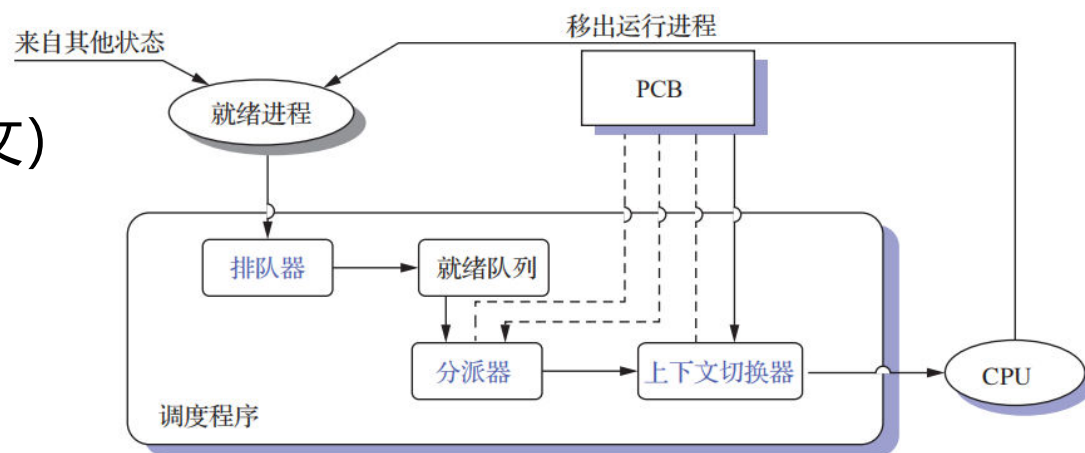
我想想





## 进程调度的任务

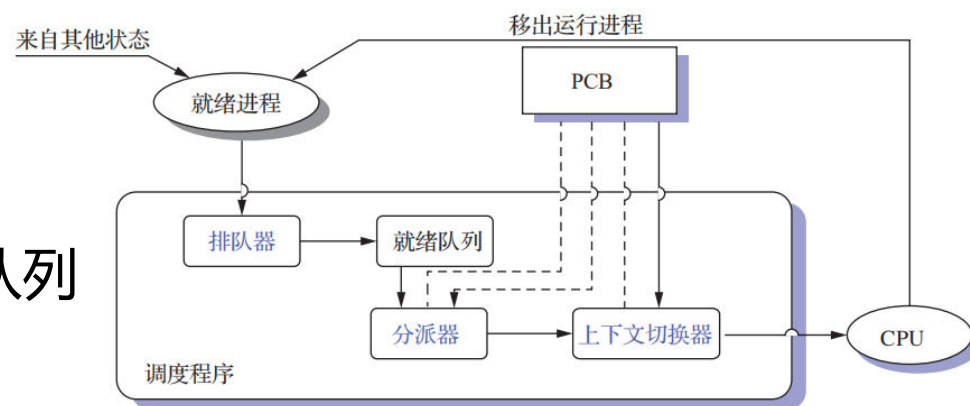
- 保存处理机的现场信息（上下文）
  - ❖ 程序计数器、通用寄存器等
  - ❖ 保存在那？
- 按某种算法选取进程
  - ❖ 从就绪队列按照某种算法选取进程
  - ❖ PCB组织方式（线性、链接、索引）
- 把处理器分配给进程
  - ❖ PCB中CPU上下文→CPU各个寄存器中（硬件）





## 进程调度机制（调度程序分为三部分）

- 排队器：用于将就绪进程插入相应的就绪队列
- 分派器：用于将选定的进程移出就绪队列
- 上下文切换器：进行新旧进程之间的上下文切换
  - ❖ 旧进程CPU上下文→进程PCB中CPU上下文，装入分派程序上下文运行分派程序
  - ❖ 移出分派程序上下文，装入新进程PCB中CPU上下文





# 进程调度的方式



## 非抢占方式:

一旦把处理机分配给某进程后, 便让该进程一直执行, 直至该进程完成或发生某事件而被阻塞时, 才再把处理机分配给其他进程, **决不允许某进程抢占已经分配出去的处理机。**

(批处理系统, 分时, 实时)



## 抢占方式:

允许调度程序根据某种原则, 去暂停某个正在执行的进程, 将已分配给该进程的处理机重新分配给另一进程。 **(现代OS广泛采用)**

- **优先权原则**: 允许优先权高的新到进程抢占当前进程的处理机
- **短作业优先原则**: 短作业可以抢占当前较长作业的处理机
- **时间片原则**: 各进程按时间片运行, 当一个时间片用完后, 便停止该进程的执行而重新进行调度

分时: 只有采用抢占调度方式才能实现人机交互

实时: 抢占调度方式能满足实时任务需求



# 处理机调度

- 取决于操作系统的类型和目标
- 不同的操作系统，有不同的调度方式和算法
- 有面向用户的准则，也有面向系统的准则

## 处理机调度算法的目标

### 如何判断调度算法的好坏-幼儿园老师给孩子喂饭的过程

- 前提：孩子较小，需要老师协助进食；老师数量较少，不可能一对一；孩子要在有限的时间完成进餐
- 法一：孩子沿饭桌围成一圈，老师按座位顺序逐个喂食小孩，每人一口，喂完一圈，从头继续；
- 法二：孩子排成一列，一个小孩喂完再喂下一个；（排在后面的会长时间饥饿）而且，老师喂完一口饭后，孩子有咀嚼、吞咽的过程，老师必须等待；
- 法三：不按顺序，谁吃的快先喂谁；
- 法四：按饭量由小到大的顺序喂饭；

• .....





# 处理机调度算法的目标



## 共同目标:

- 资源利用率  
(保持忙碌)
- 公平性  
(进程饥饿)
- 平衡性  
(CPU/IO繁忙)
- 策略强制执行  
(安全策略等)

$$\text{CPU利用率} = \frac{\text{CPU有效工作时间}}{\text{CPU有效工作时间} + \text{CPU空闲等待时间}}$$



## 批处理系统的目标:

- 平均周转时间短、系统吞吐量高、处理机利用率高



## 分时系统的目标:

- 响应时间快、均衡性



## 实时系统的目标:

- 截止时间的保证、可预测性



## 评价指标

### 批处理系统的目标

**周转时间**：作业完成时刻-作业到达时刻（用户）

等待作业调度时间+等待进程调度时间+进程执行时间+进程等待IO完成时间

**平均周转时间**：作业周转总时间/作业个数（提供系统资源利用率的同时使大多数用户满意）

$$T = \frac{1}{n} \left( \sum_{i=1}^n T_i \right)$$

**带权周转时间**：周转时间/服务时间

**平均带权周转时间**：带权周转总时间/作业个数

$$W = \frac{1}{n} \left( \sum_{i=1}^n \frac{T_i}{T_{s_i}} \right)$$



# 评价指标



吞吐量：

- 单位时间内所完成的作业数



等待时间（进程调度）：

- 进程在就绪队列中等待调度的所有时间之和。



## 分时系统的目标



响应时间:

- 从用户通过键盘提交请求开始, 直到系统首次显示出处理结果为止的一段时间。



响应时间包括:

- 从键盘输入的请求信息传送到处理机的时间
- 处理机对请求信息进行处理的时间
- 将所形成的响应回送到终端显示器的时间



均衡性 (复杂的任务, 其相应时间可以稍长)

- 响应时间快慢与用户请求复杂度相适应



## 实时系统的目标



截止时间:

- 是指某任务必须开始执行的最迟时间, 或必须完成的最迟时间










保证请求的可预测性:

- 用户请求可预知 (用户在看电影, 第 $i$ 时刻请求第 $i$ 帧, 那么第 $i+1$ 时刻很大可能会请求第 $i+1$ 帧)



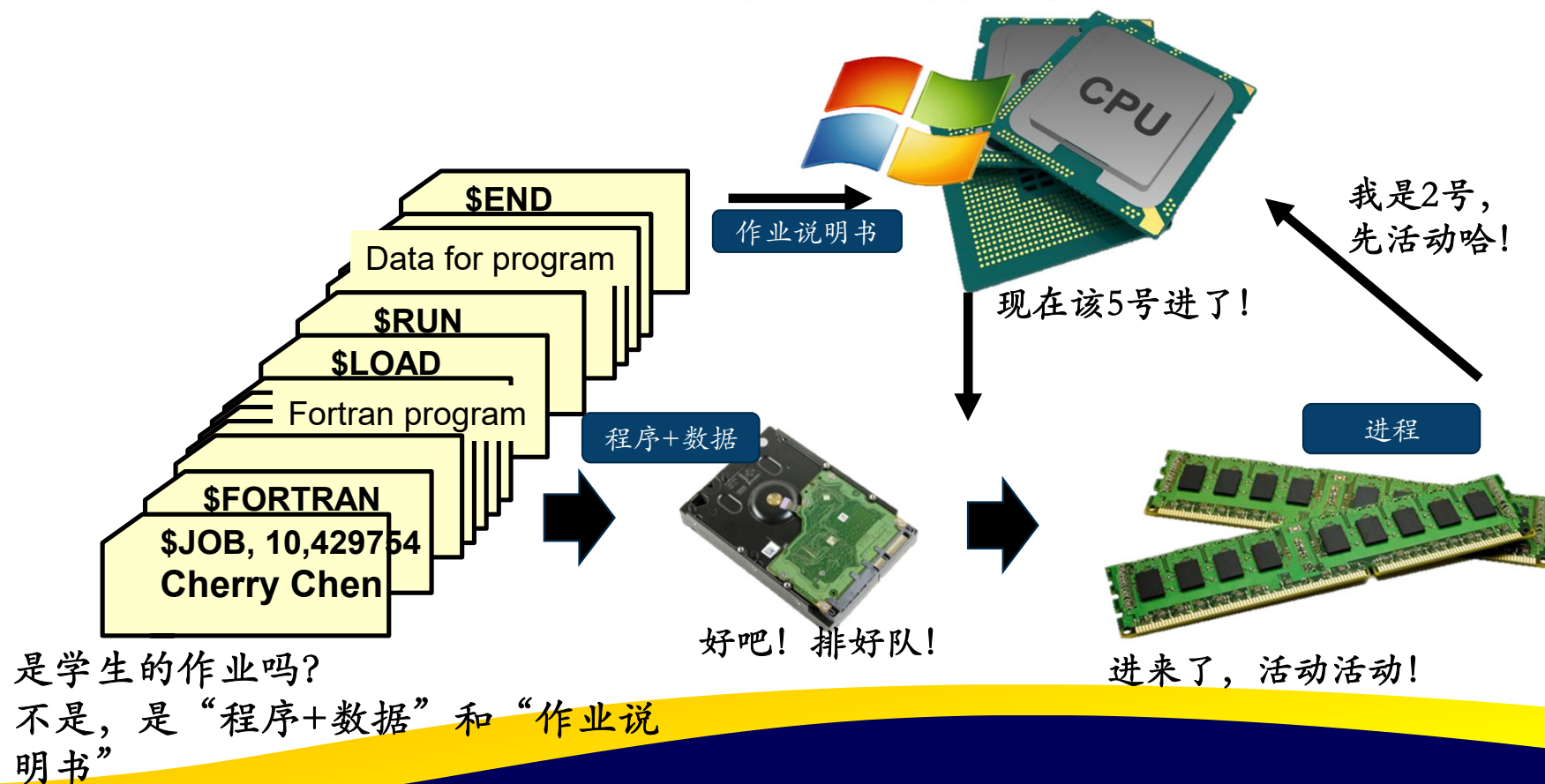
## 内容导航:

-  3.1 处理机调度概述
-  **3.2 调度算法**
-  3.3 实时调度
-  3.4 Linux进程调度
-  3.5 死锁概述
-  3.6 预防死锁
-  3.7 避免死锁
-  3.8 死锁的检测与解除

# 第3章 处理机调度与死锁

---

**作业：**在多道批处理操作系统中，用户提交给系统的一项相对独立的工作





# 作业调度

处理机系统中的作业

作业调度的主要任务

作业调度算法

## 作业和作业步

1. **作业 (JOB)**：是用户在一次算题过程中或一次事务处理中，要求计算机系统所做的工作的集合。在批处理系统中，以作业为单位从**外存**调入内存
2. **作业和进程的关系**：作业是比进程更广泛的概念，不仅包含了通常的程序和数据，而且还配有一份**作业说明书**，系统根据作业说明书对程序运行进行控制



# 作业调度

作业步: 作业运行期间, 每个作业由若干相互独立又相互关联的顺序加工步骤才能得到结果。每个加工步骤称为一个作业步

处理机系统中的作业

作业调度的主要任务

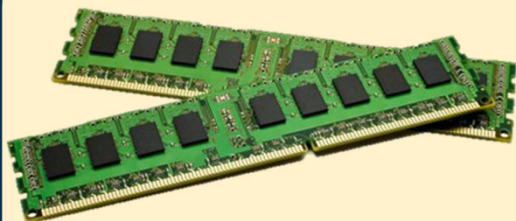
作业调度算法



“程序+数据”：  
(EXE文件、动态链接库文件) + 数据文件  
(iexplore.exe, ieproxy.dll) +  
baidu.com.html



- ①编译=让iexplore他们变成机器码;
- ②装入=给iexplore他们分配地盘;
- ③链接=把iexplore和ieproxy嫁接。



iexplore.exe  
(baidu.com.html)

ieproxy.dll

处理机系统中的作业

作业调度的主要任务

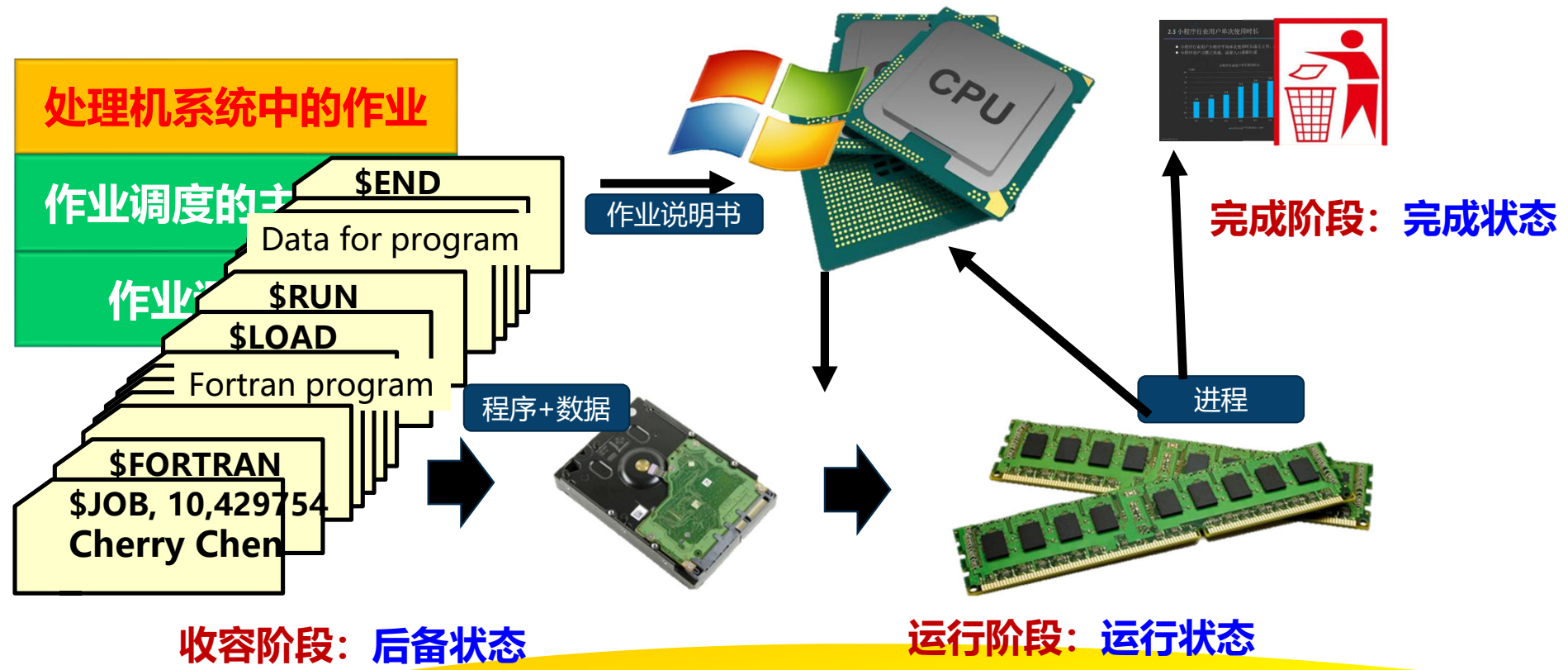
作业调度算法

作业名	
资源要求	预估的运行时间 最迟完成时间 要求的内存量 要求外设类型、台数 要求的文件量和输出量
资源使用情况	进入系统时间 开始运行时间 已运行时间 内存地址 外设台号
类型级别	控制方式 作业类型 优先级
状态	
用户账户……	

## 作业控制块

- 作业控制块 (JCB)**: 多道批处理系统中, 为每个作业设置一个作业控制块。 JCB是一个作业在系统中存在的惟一标志, 系统根据JCB才感知到作业的存在
- JCB包含内容**: 作业控制块JCB中包含了对作业进行管理的必要信息, JCB中的信息一部分是从用户提供的作业控制卡或作业说明书中得到, 另一部分是记录作业运行过程中的动态信息
- JCB的生成**: 作业提交给系统后, 便由“作业注册”程序**为作业**建立一个作业控制块JCB, 放入后备队列

## 作业运行的三个阶段和三种状态



# 作业调度



处理机系统中的作业

作业调度的主要任务

作业调度算法

## (A) 接纳多少个作业

- 取决于多道程序度，即允许多少个作业同时在内存中运行。
- 作业太少 资源利用率低
- 作业太多 服务质量下降

## (B) 接纳哪些作业：作业调度算法

- 先来先服务
- 短作业优先
- 优先权高优先
- 高响应比优先

1、根据JCB，审查系统能否满足用户作业的资源需求

3、为它们创建进程、分配必要资源

2、按一定算法，从外存后备队列中选取某些作业调入内存

4、将新创建的进程插入就绪队列，准备执行



- 先来先服务调度算法(FCFS)
- 短作业优先调度算法(SJF)
- 优先级调度算法(PR)
- 高响应比优先调度算法(HRRN)

---

FCFS、SJF、PR既可用于作业调度, 也可用于进程调度

---

## 进程调度的任务、机制

### 进程调度方式

### 进程调度算法

## 进程调度的任务

1. **保存处理机现场信息。** 进程调度程序把当前进程的现场信息，如程序计数器及通用寄存器的内容等保留在该进程PCB的现场信息区中
2. **按某种算法挑选进程。** 根据一定的**调度算法**(如优先级算法)，从就绪队列中选出一个进程来，并把它状态改为运行态，准备把CPU分配给它
3. **把处理器分配给进程。** 为选中的进程恢复现场信息，并把CPU的控制权交给该进程，它接着上次间断的地方继续运行



# 进程调度

## 进程调度的任务、机制

进程调度方式

进程调度算法

## 进程调度机制

1. **排队器**: 将就绪进程按一定方式排成队列
2. **分派器（分派程序）**: 把进程调度程序选定的进程，从就绪队列中取出，进行上下文切换，把处理器分配给它
3. **上下文切换机制**: ①保存当前进程上下文，装入分派程序上下文，分派程序运行；  
②移出分派程序，装入新选进程上下文

进程调度的任务、机制

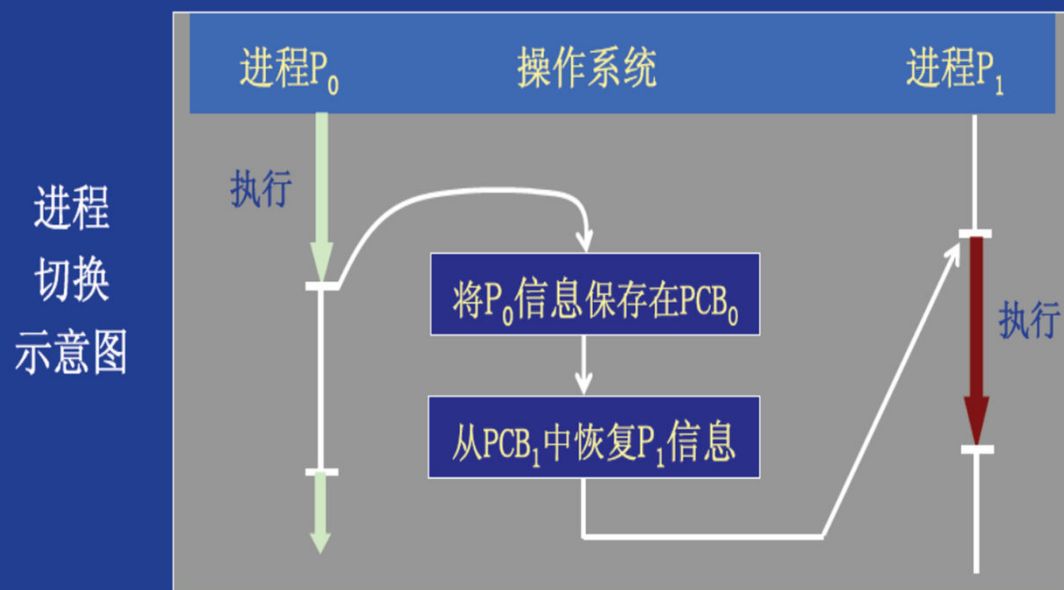
进程调度方式

进程调度算法

## 进程调度机制

➤分派程序的主要功能：

1. 进行进程切换；
2. 转到用户态；
3. 开始执行被选中的进程。







# 进程调度算法



先来先服务调度算法(FCFS)



短作业优先调度算法(SJF)



优先权调度算法(PR)



时间片轮转调度算法(RR)



多级队列调度算法



多级反馈队列调度算法



基于公平原则的调度算法

01

按照作业到达的先后次序来进行调度

作业	运行时间
J1	24
J2	3
J3	3

02

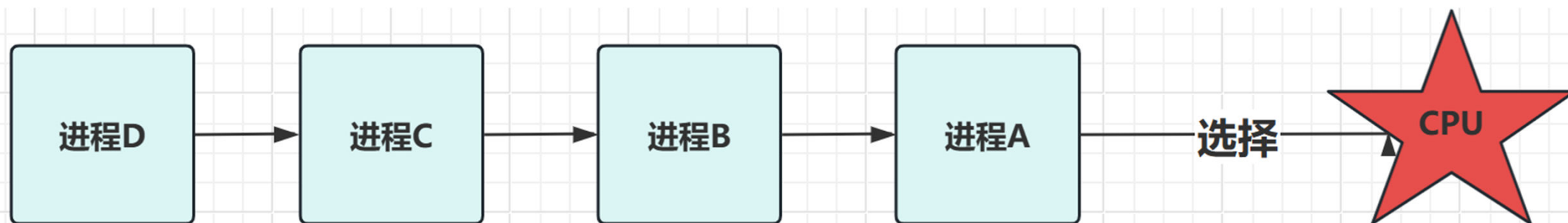
假定作业(“同时”)到达顺序如下:  
J1, J2, J3 该调度的甘特图(Gantt)  
为:



➤ 平均等待时间 =  $(0 + 24 + 27)/3 = 17$       ➤ 平均周转时间 =  $(24 + 27 + 30)/3 = 27$



## 先来先服务 (FCFS) 调度算法



该图片来自网络[https://blog.csdn.net/qq\\_64680177/article/details/134777140](https://blog.csdn.net/qq_64680177/article/details/134777140)

01

假定进程到达顺序如下

P<sub>2</sub> , P<sub>3</sub> , P<sub>1</sub> (J<sub>2</sub> , J<sub>3</sub> , J<sub>1</sub>) .

02

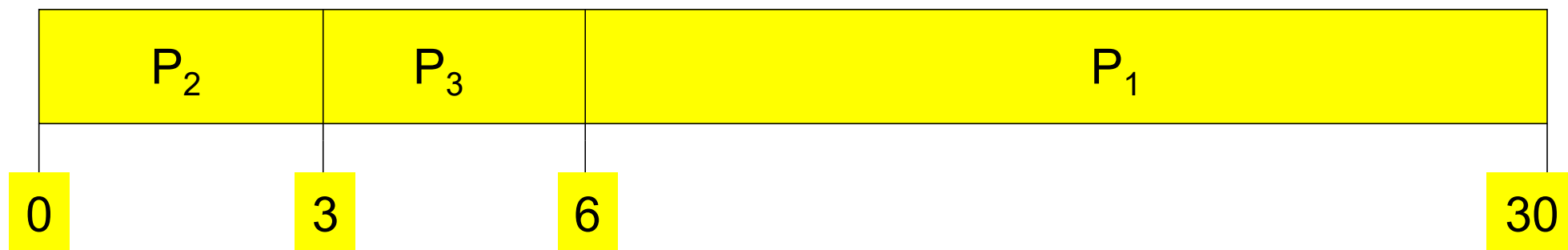
该调度的Gantt图为：

03

比前例好得多

04

此结果产生是由于短进程先于长进程到达



➤ 平均等待时间 =  $(6 + 0 + 3)/3 = 3$

➤ 平均周转时间 =  $(30 + 3 + 6)/3 = 13$



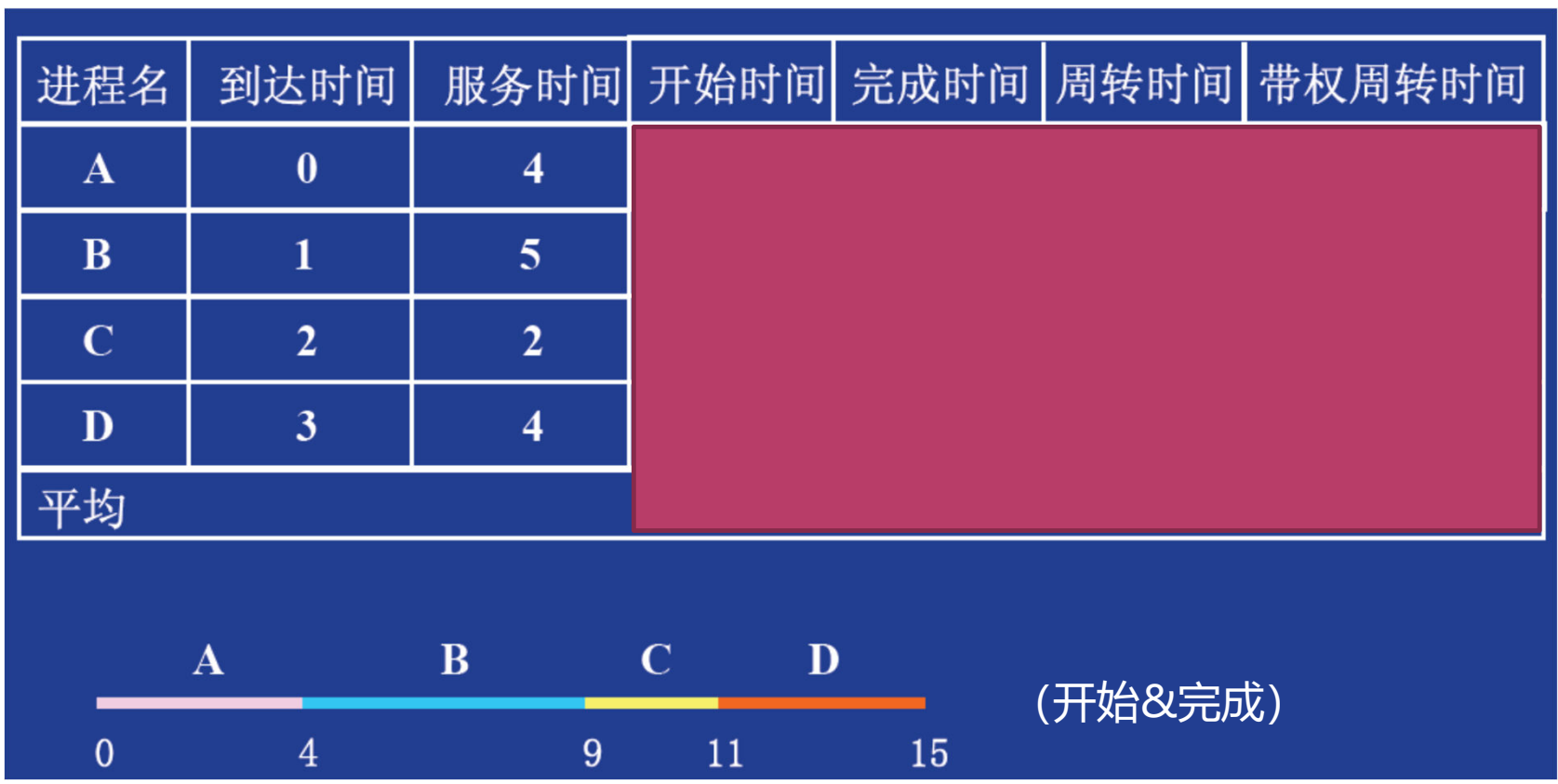
### 先来先服务调度算法(FCFS)的特点

1. 作业调度和进程调度均可，最简单，本质上属**非抢占方式**（**优先**考虑在系统中等待时间最长的J or P)
2. 有利于**长作业/进程**，不利于**短作业**
3. 有利于**CPU繁忙型**的作业(如通常的科学计算)，而不利于**I / O繁忙**的作业/进程（如大多数的事务处理）
4. 现在很少单独使用，而是结合其他算法（优先级算法）使用



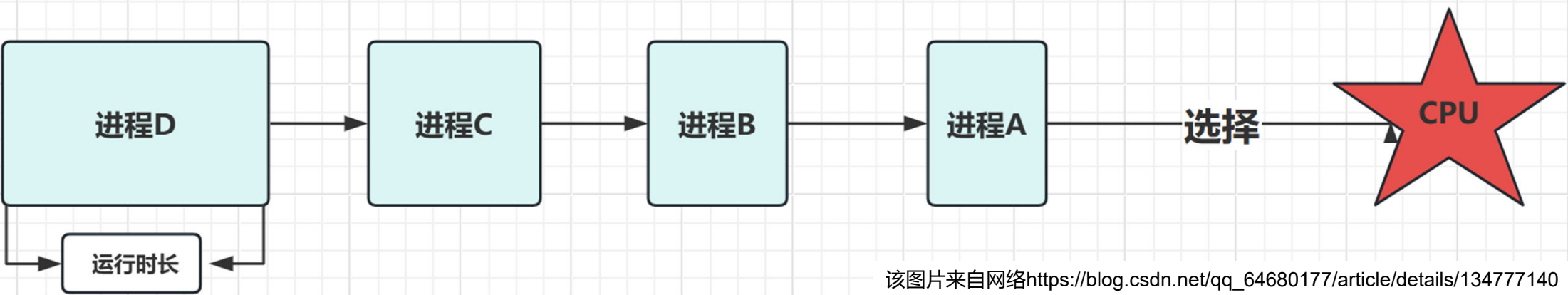
# FCFS调度算法

例





# 短作业优先 (SJF) 调度算法



该图片来自网络[https://blog.csdn.net/qq\\_64680177/article/details/134777140](https://blog.csdn.net/qq_64680177/article/details/134777140)



## 短作业优先 (SJF) 调度算法(1)



SJF算法：既可用于作业，也可用于进程

- 对作业：从**后备队列**中选择若干个**估计运行时间最短**的作业。
- 对进程：关联到每个进程下次运行的CPU区间长度，调度最短的进程。



对进程调度，SJF有两种模式：

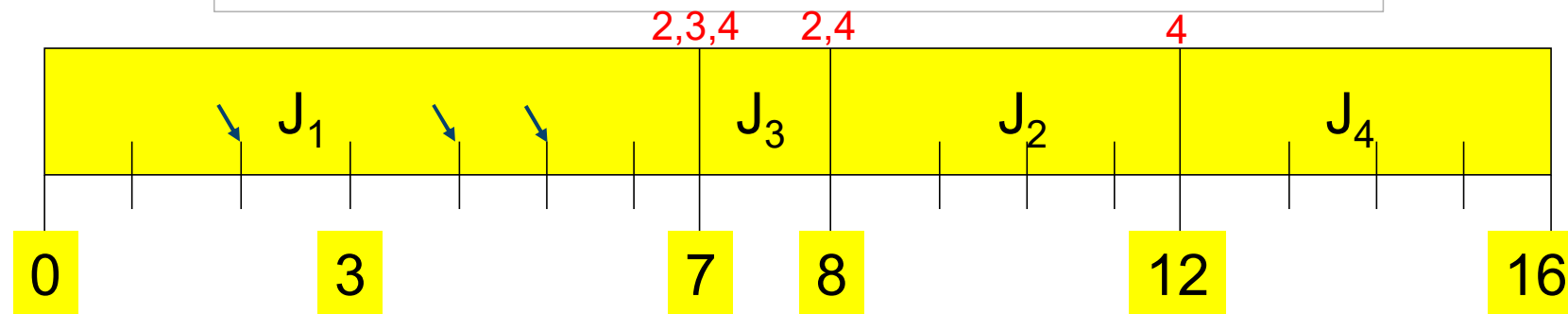
- **非抢占式SJF**
- **抢占式SJF (来了个更短的进程)**：抢占发生在有比当前进程剩余时间片更短的进程到达时，也称为最短剩余时间优先调度



SJF是最优的（对一组指定的进程而言），它给出了最短的平均等待时间。



作业	到达时间	运行时间
$J_1$	0.0	7
$J_2$	2.0	4
$J_3$	4.0	1
$J_4$	5.0	4



➤ 平均等待时间 =  $(0 + 6 + 3 + 7)/4 = 4$

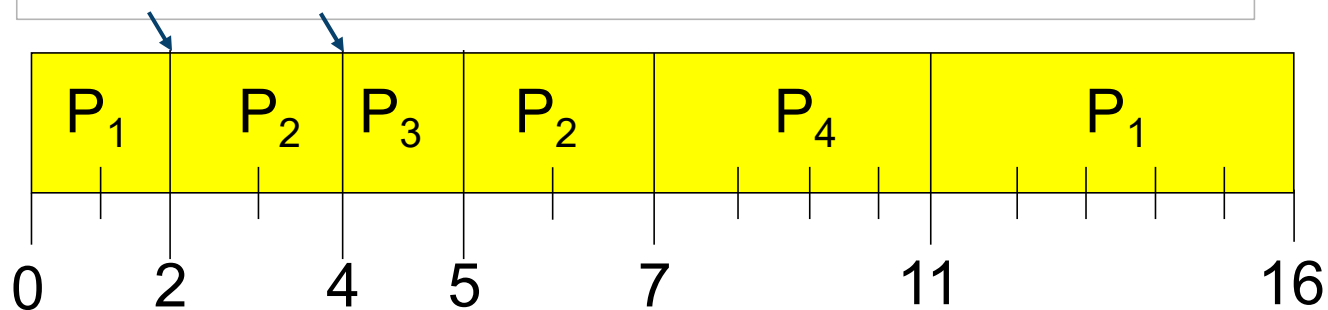
➤ 平均周转时间 =  $(7 + 10 + 4 + 11)/4 = 8$



## 抢占式SJF举例

例:

<u>进程</u>	<u>到达时间</u>	<u>区间时间</u>
P1	0	7
P2	2	4
P3	4	1
P4	5	4



➤ 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) 调度算法 (2)

SJF比FCFS算法有明显改进

### 缺点:

-  只能估算进程的运行时间 (估值不准确, 作业说明书, 偏长估计), 所以通常用于作业调度
-  对长作业不利 (周转时间明显增加, SJF完全忽视等待时间→进程饥饿发生)
-  采用SJF算法时, 人-机无法实现交互
-  完全未考虑作业的紧迫程度 (优先级)



## 短作业优先 (SJF) 调度算法 (3)

调度算法 \ 作业情况	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 (a)	完成时间	4	7	12	14	18	
	周转时间	4	6	10	11	14	9
	带权周转时间	1	2	2	5.5	3.5	2.8
SJF (b)	完成时间	4	9	18	6	13	
	周转时间	4	8	16	3	9	8
	带权周转时间	1	2.67	3.1	1.5	2.25	2.1



## 优先级调度算法PR (1)



既可用于作业调度，也可用于进程调度。



基于作业/进程的紧迫程度，由外部赋予作业相应的优先级，调度算法根据优先级进行调度。

- 每个进程都有一个优先数，优先数为整数。
- 默认：小的优先数具有高优先级。
- 目前主流的操作系统调度算法。



高响应比优先调度算法是一种优先级调度算法，用于作业调度。



# 进程优先级调度算法



## 优先级调度算法的类型

类比，军人、孕妇等优先级高

- 非抢占式
- 抢占式



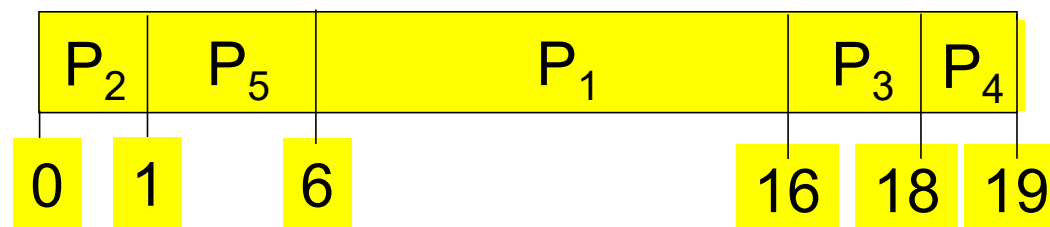
## 优先级类型

- 静态优先级
  - ❑ 创建进程时确定优先数(整数)，在进程的整个运行期间保持不变
  - ❑ 简单易行，系统开销小
  - ❑ 不够精确，可能会出现优先级低的进程长期没有被调度的情况
- 动态优先级
  - ❑ 创建进程时先赋予其一个优先级，然后其值随进程的推进或等待时间的增加而改变



## 非抢占式优先级调度算法举例

进程	优先级	运行时间
P1	3	10
P2	1	1
P3	3	2
P4	4	1
P5	2	5



- 平均等待时间 =  $(6 + 0 + 16 + 18 + 1) / 5 = 8.2$
- 平均周转时间 =  $(16 + 1 + 18 + 19 + 6) / 5 = 12$



## 优先级调度算法的优缺点

### 优点

- 实现简单，考虑了进程的紧迫程度
- 灵活，可模拟其它算法  
(FCFS：等待时间；SJF：运行时间)

### 存在问题

- 饥饿：低优先级的进程可能永远得不到运行

### 解决方法

- 老化：视进程等待时间的延长提高其优先数



## OS 高响应比优先调度算法 (PR的特例)

➤ 既考虑作业的等待时间，又考虑作业的运行时间 (FCFS;SJF)

➤ 优先级： 
$$\text{优先级} = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}}$$

➤ 响应比： 
$$R_p = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}} = \frac{\text{响应时间}}{\text{要求服务时间}}$$

响应时间：从用户通过键盘提交请求开始，直到系统首次显示出处理结果为止的一段时间。

- 如等待时间相同，取决于运行时间，运行时间越短，优先级越高，类似于SJF
- 如运行时间相同，取决于等待时间，类似于FCFS
- 长作业可随其等待时间的增加而提高，也可得到服务
- 缺点：每次调度之前，都需要计算响应比，增加系统开销

# OS 高相应比优先调度算法 (PR的特例)

三个作业在一台处理机上单道运行，  
9:40 进行作业调度，问三个作业的执行次序？

9:40 调度时：

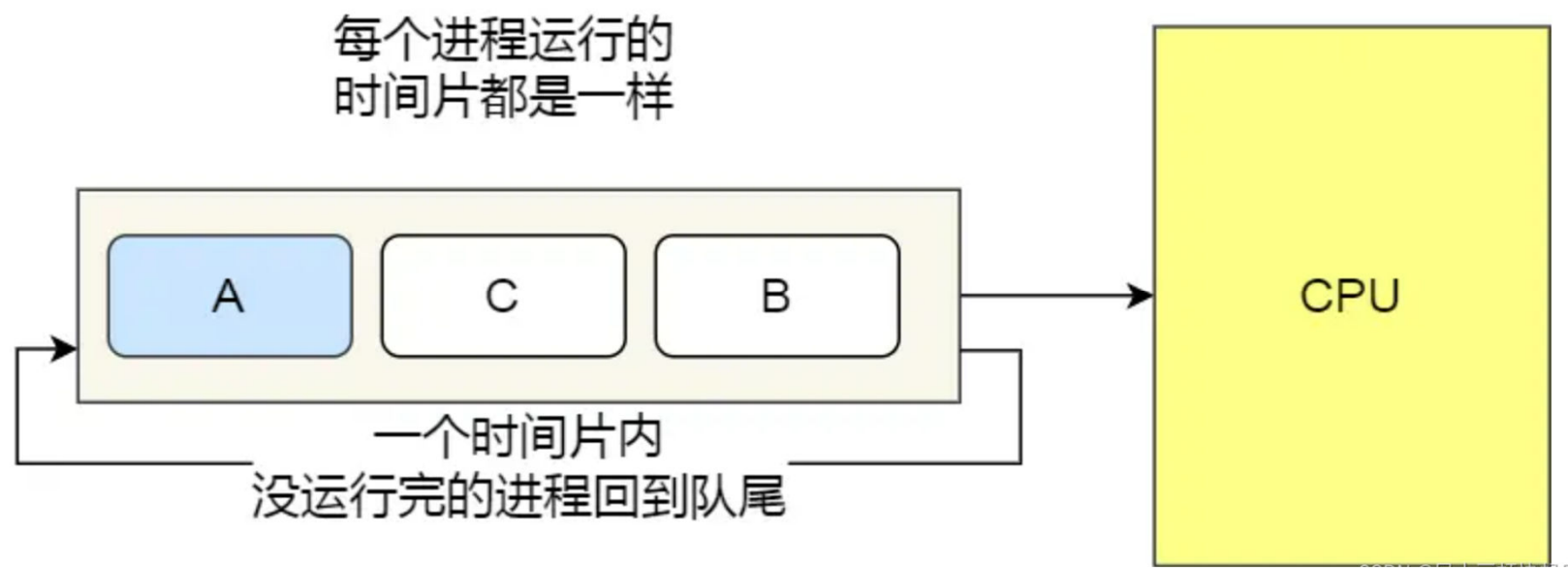
作业名	到达时间	服务时间
J1	8:00	2小时
J2	8:30	1小时
J3	9:30	0.25小时

10:40 (J2完成) 调度时：

执行次序：J2、J3、J1



# 时间片轮转(RR)调度算法



该图片来自网络[https://blog.csdn.net/qq\\_64680177/article/details/134777140](https://blog.csdn.net/qq_64680177/article/details/134777140)



专为分时系统设计，类似于FCFS，但增加了抢占



时间片  
quantum

小单位的CPU时间，通常为10~100毫秒（不同系统不同）



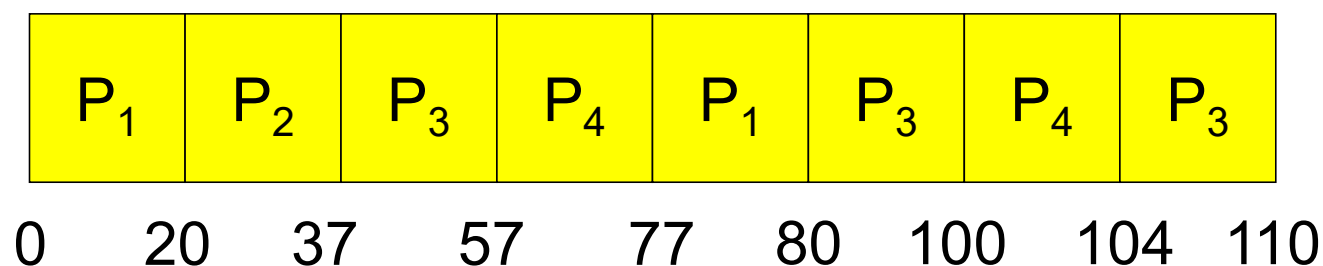
为每个进程分配不超过一个时间片的CPU。时间片用完后，该进程将被抢占并插入就绪队列末尾，循环执行



假定就绪队列中有 $n$ 个进程、时间片为 $q$ ，则每个进程每次得到 $1/n$ 的、不超过 $q$ 单位的成块CPU时间，没有任何一个进程的等待时间会超过 $(n-1)q$ 单位

进程	运行时间
P1	23
P2	17
P3	46
P4	24

➤ Gantt图如下:



- 平均等待时间:  $(57+20+64+80)/4 = 55.25$
- 平均响应时间 (下一页):  $(0+20+37+57)/4 = 28.5$
- 通常, RR的平均周转时间比SJF长, 但响应时间要短一些.



**(进程) 响应时间 (Response time)** : The time spent when the process is in the ready state and gets the CPU for the first time. (自进程就绪至进程第一次获得CPU响应的时间)

**(进程) 响应时间 (Response time) = 第一次响应 - 到达时间**

(作业) 响应时间: 从用户通过键盘提交请求开始, 直到系统首次显示出处理结果为止的一段时间。



## 特性

- $q$  大  $\rightarrow$  FCFS
- $q$  小  $\rightarrow$  增加上下文切换的时间, 效率降低

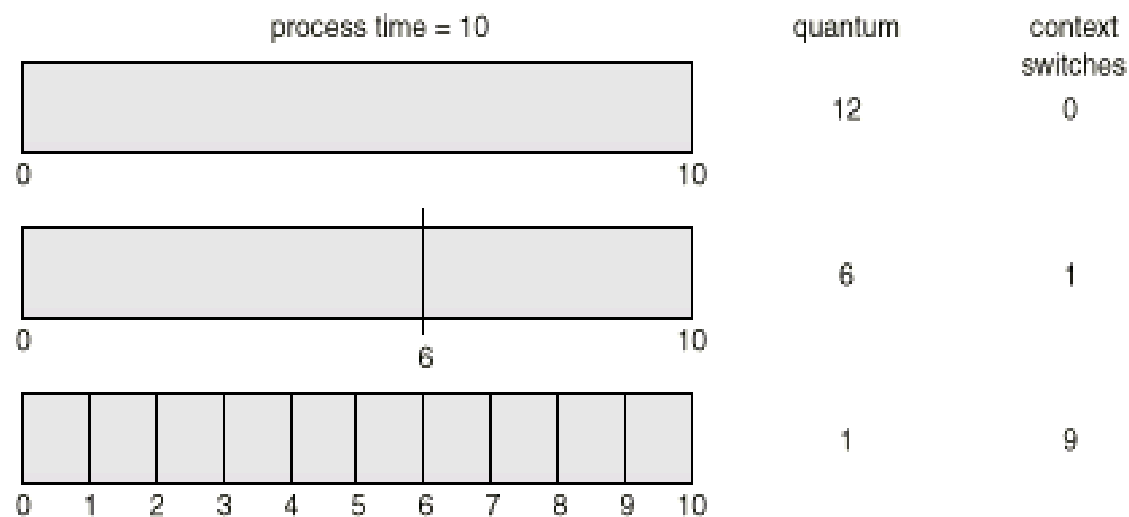


## 时间片设置应考虑

- 系统对响应时间的要求
- 就绪队列中进程的数目
- 系统的处理能力



**一般准则:** 时间片/10 > 进程上下文切换时间





## RR例子

作业情况 时间片	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	4	2	4	
RR q=1	完成时间	15	12	16	9	17	
	周转时间	15	11	14	6	13	11.8
	带权周转时间	3.75	3.67	3.5	3	3.25	3.43
RR q=4	完成时间	4	7	11	13	17	
	周转时间	4	6	9	10	13	8.4
	带权周转时间	1	2	2.25	5	3.25	2.7





# 多级队列调度算法



就绪队列从一个分为多个（按性质），如：

- 前台[交互式]
- 后台[批处理]



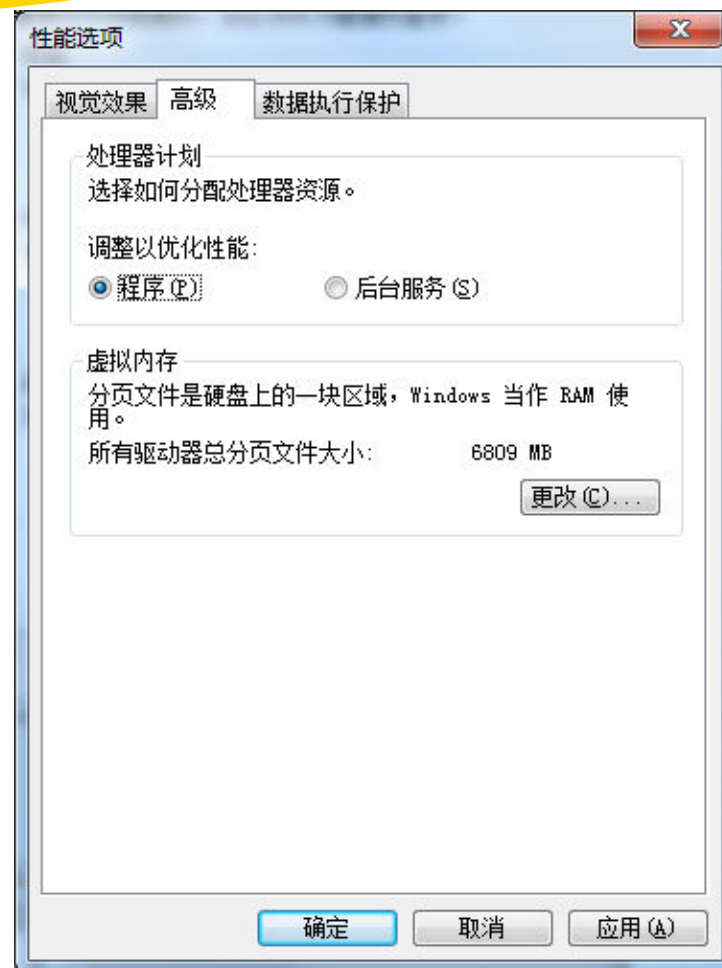
每个队列有自己的调度算法

- 前台 – RR
- 后台 – FCFS



调度须在队列间进行（队列间有优先级）

- 固定优先级调度，即前台运行完后再运行后台，有可能产生饥饿。
- 给定时间片调度，即每个队列得到一定的CPU时间，进程在给定时间内执行；如80%的时间执行前台的RR调度，20%的时间执行后台的FCFS调度



主要应用于多处理机系统



# 多级反馈队列调度算法



进程能在不同的队列间移动



其他调度算法的局限性

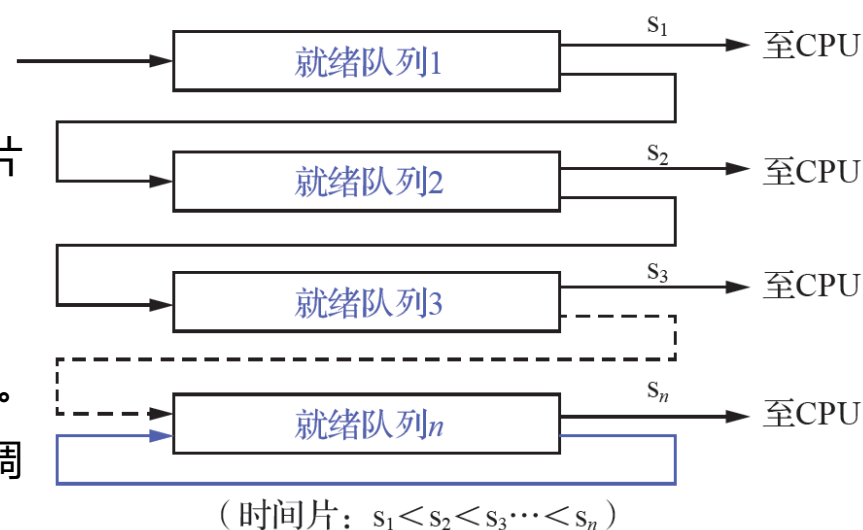
- 短进程优先的调度算法，仅照顾了短进程而忽略了长进程
- 如果并未指明进程的长度，则短进程优先和基于进程长度的抢占式调度算法都将无法使用。



多级反馈队列调度算法的优点：

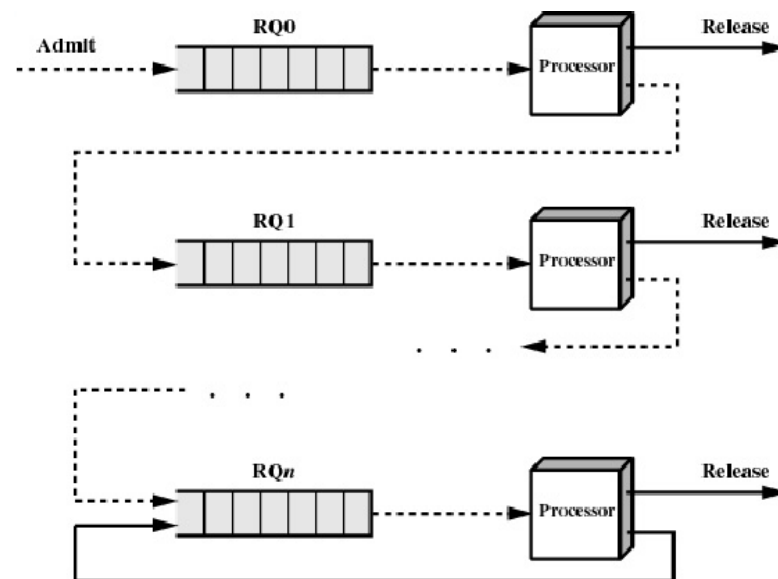
- 不必事先知道各种进程所需的执行时间；
- 可以满足各种类型进程的需要。

- 按照FCFS原则排序，设置N个就绪队列为Q1,Q2,...QN，每个队列中都可以放很多作业；
- 为这N个就绪队列赋予不同的优先级，第一个队列的优先级最高，第二个队列次之，其余各队列的优先权逐个降低；
- 设置每个就绪队列的时间片，优先权越高，算法赋予队列的时间片越小。时间片大小的设定按照实际作业（进程）的需要调整(1,2,4,...)；
- 进程在进入待调度的队列等待时，首先进入优先级最高的Q1等待。
- 首先调度优先级高的队列中的进程。若高优先级中队列中已没有调度的进程，则调度次优先级队列中的进程。
- 对于同一个队列中的各个进程，按照RR调度。
- 在低优先级的队列中的进程在运行时，又有新到达的作业，那么在运行完这个时间片后，CPU马上分配给新到达的作业即抢占式调度CPU。



例如，若一个进程总共需运行100个时间片

1. 初始时指定它在优先级最高的进程组中，很快就会在CPU上运行一个时间片，之后优先级也降低一个级别
2. 当它第二次有机会在CPU上运行时，它将运行 $2t$
3. 以后它将在CPU上运行的时间长度依次是4, 8, 16, 32和64个 $t$ ，最后一次运行时，只须64个 $t$ 中37个 $t$ 就可完成
4. **总共需调度7次**。比较单纯的轮转法，节省了93次切换时间





## 多级反馈队列调度例子

假设系统中有3个就绪队列Q1、Q2、Q3，时间片分别为2、4、8。

现在有3个作业J1、J2、J3分别在时间 0、1、3时刻到达。而它们所需要的CPU时间分别是3、2、1个时间片。请使用多级反馈队列调度分析调度过程。



## 多级反馈队列调度

**时刻0:** J1到达。于是进入到Q1，运行1个时间片。

**时刻1:** J2到达。由于时间片仍然由J1掌控，于是等待。J1在运行了1个时间片后，已经完成了在Q1中的2个时间片的限制，于是J1置于Q2队列等待被调度。现在处理机分配给J2。

**时刻2:** J1进入Q2队列等待调度，J2获得CPU开始运行。

**时刻3:** J3到达，由于J2的时间片未到，故J3在Q1队列等待调度，J1也在Q2队列等待调度。

**时刻4:** J2处理完成，由于J3、J1都在等待调度，但是J3所在的队列比J1所在的队列的优先级要高，于是J3被调度，J1继续在Q2队列等待。

**时刻5:** J3经过1个时间片，完成。

**时刻6:** 由于Q1队列已经空闲，于是开始调度Q2队列中的作业，则J1得到处理器开始运行。

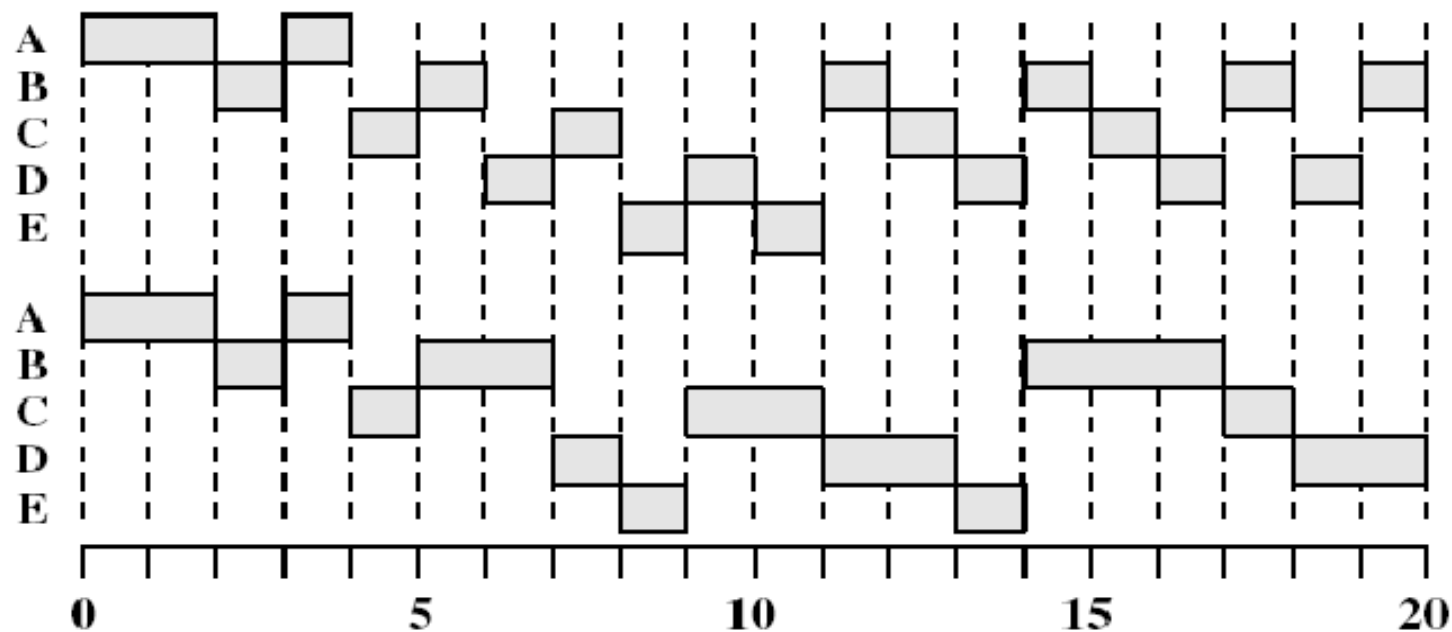
**时刻7:** J1再经过一个时间片，完成了任务。于是整个调度过程结束。

## 练习题

有下述五个进程，按照多级反馈调度算法进行调度。每个进程的到达时间和服务时间如下表所示。分别描述当时间片长度 $q=1$ 和 $q=2^i$ 时，各个时间片的执行情况，并计算每个进程的结束时间、周转时间和带权周转时间。

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

➤  $q=1$  和  $q=2^i$



FB $q = 1$	Finish Time	4	20	16	19	11	
	Turnaround Time ( $T_f$ )	4	18	12	13	3	10.00
	$T_f/T_s$	1.33	3.00	3.00	2.60	1.5	2.29
FB $q = 2^i$	Finish Time	4	17	18	20	14	
	Turnaround Time ( $T_f$ )	4	15	14	14	6	10.60
	$T_f/T_s$	1.33	2.50	3.50	2.80	3.00	2.63





### 多级反馈队列调度算法的性能

#### 1. 终端型作业用户

- ◆ 在第一队列中完成，作业短，交互型；

#### 2. 短批处理作业用户

- ◆ 周期时间较短，通常三个队列即可完成；

#### 3. 长批处理作业用户

- ◆ 依次在前  $n$  个队列中执行，然后再按轮转方式运行。



## 基于公平原则的调度算法



主要考虑调度的公平性。



保证调度算法：

- 性能保证，而非优先运行；
- 如保证处理机分配的公平性（处理机时间为 $1/n$ ）。



公平分享调度算法：

- 调度的公平性主要针对用户而言；
- 使所有用户能获得相同的处理机时间或时间比例。

**例如：**用户1： A,B,C,D; 用户2： E

为了使用户获得相同的处理机时间，强制调度算法为： A E B E C E D E A E B E C E D E



## 基于公平原则的调度算法



主要考虑调度的公平性。



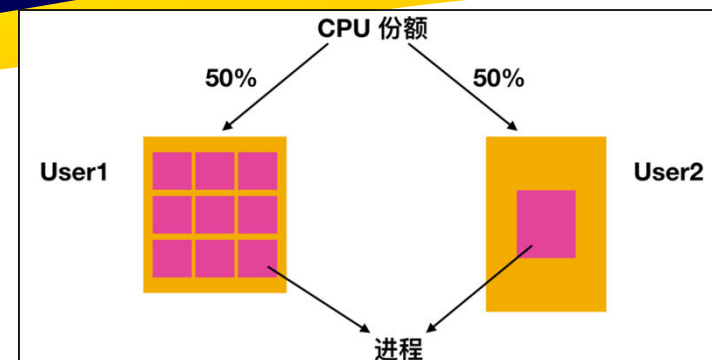
保证调度算法：

- 性能保证，而非优先运行；
- 如保证处理机分配的公平性（处理机时间为 $1/n$ ）。



公平分享调度算法：

- 调度的公平性主要针对用户而言；
- 使所有用户能获得相同的处理机时间或时间比例。



用户公平（进程不公平）

例如：用户1： A,B,C,D; 用户2： E

为了使用户获得相同的处理机时间，强制调度算法为： A E B E C E D E A E B E C E D E