

LINUX OPERATING SYSTEM

YANG

LINUX操作系统（双语）





双语课→课件内容中英混排

|Lecture 6

CPU Scheduling



本讲内容

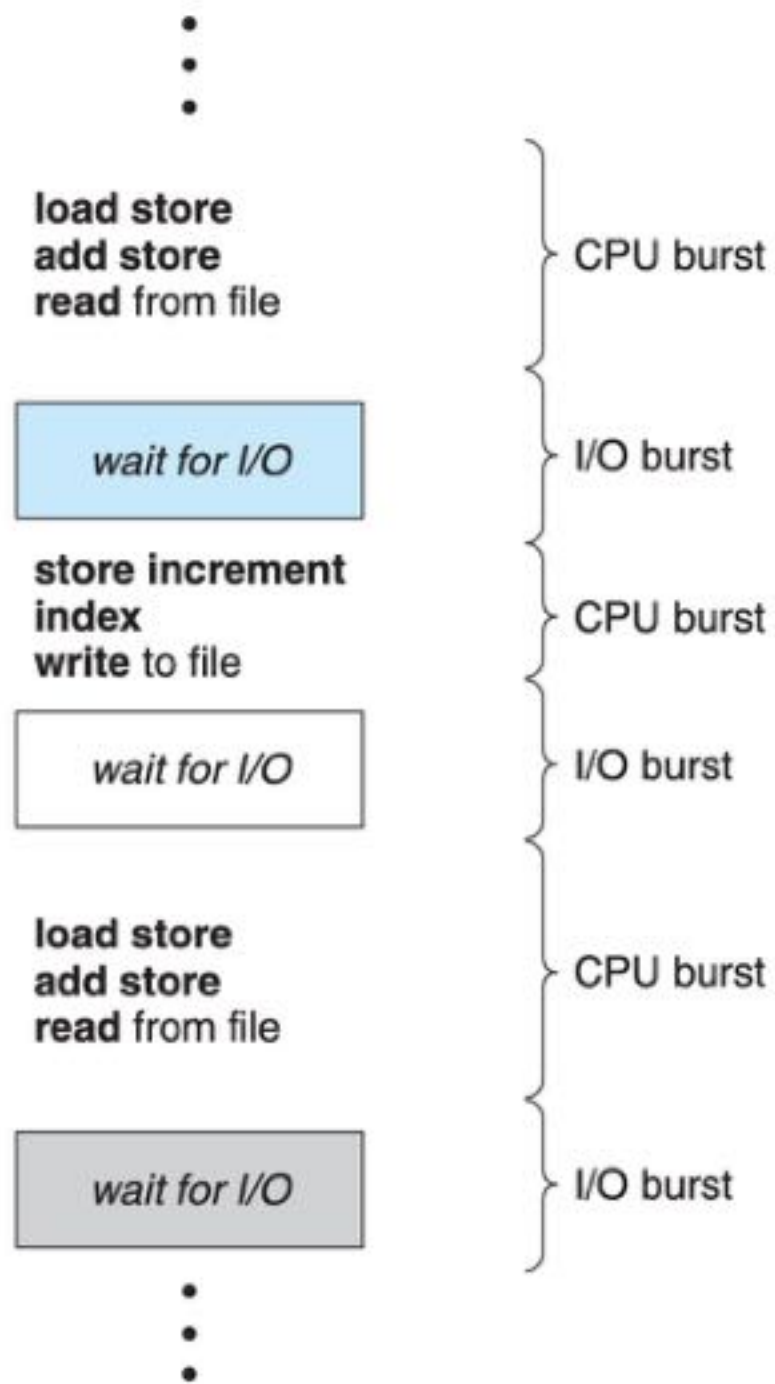
 CPU调度程序

 CPU调度准则

 调度算法

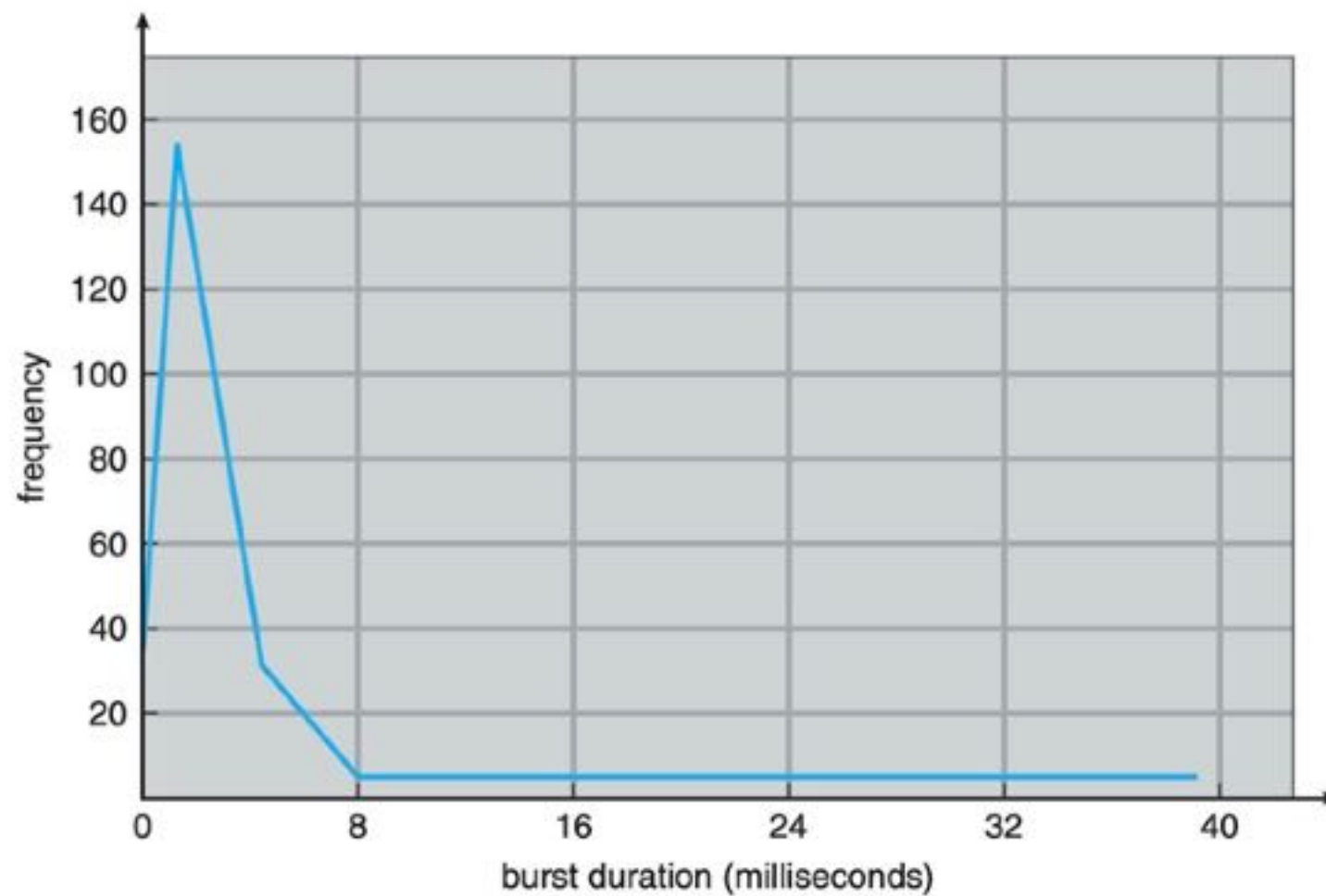
CPU调度程序

基本概念



- 💡 多道程序设计的目的将CPU的利用率最大化。
- 💡 多个进程同时存在于内存（并发），当一个进程暂不使用CPU时，系统调度另一个进程占用CPU。

CPU-BURST DURATIONS



CPU-bound program




I/O-bound program

CPU调度程序

- 🧠 Whenever the CPU becomes **idle**, the operating system must select one of the processes in the **ready queue** to be executed. The selection process is carried out by the **CPU scheduler**.

抢占调度

非抢占调度 (Nonpreemptive scheduling)

 一旦某个进程得到CPU，就会一直占用到终止或等待状态。

抢占调度 (Preemptive scheduling)

CPU调度准则

调度算法性能的衡量

- 🧠 CPU利用率：CPU的忙碌程度
- 🧠 响应时间：从提交任务到第一次响应的的时间
- 🧠 等待时间：进程累积在就绪队列中等待的时间
- 🧠 周转时间：从提交到完成的时间
- 🧠 吞吐率：每个时钟单位处理的任务数
- 🧠 公平性：以合理的方式让各个进程共享CPU

调度性能指标

- 🧠 作业 (job) = 进程 (process)
- 🧠 假设作业*i*提交给系统的时刻是 t_s ，完成的时刻是 t_f ，所需运行时间为 t_k ，那么：
- 🧠 **平均作业周转时间 T** (t_i 是单个作业的周转时间)

$$T = \left(\sum_{i=1}^n t_i \right) \times \frac{1}{n} \quad (t_i = t_f - t_s)$$

调度算法

先来先服务(FCFS)

👤 First-Come, First-Served (FCFS)

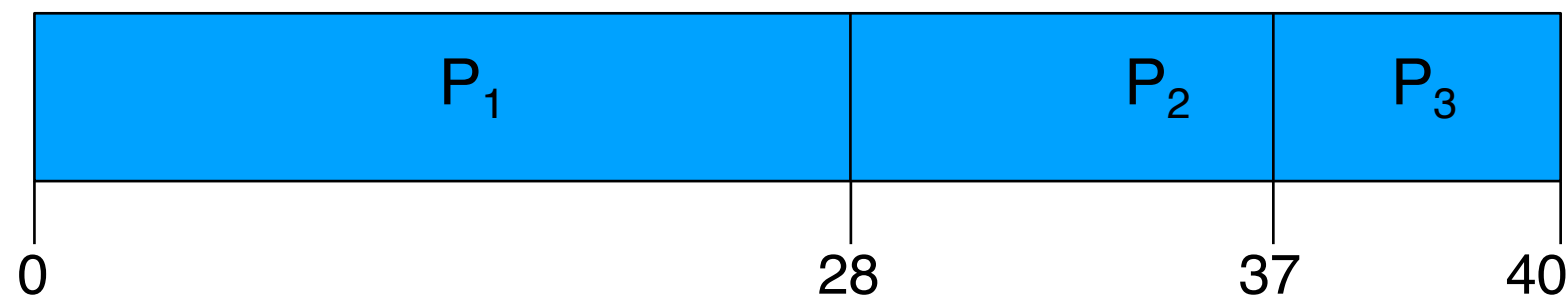
👤 早期系统里，FCFS意味着一个程序会一直运行到结束(尽管其中会出现等待I/O的情况)

👤 如今，当一个程序阻塞时会让出CPU

👤 例题:

Process	Time
P1	28
P2	9
P3	3

👤 如果三个进程的到达顺序是: P1 , P2 , P3



👤 等待时间分别是: $P1 = 0$; $P2 = 28$; $P3 = 37$

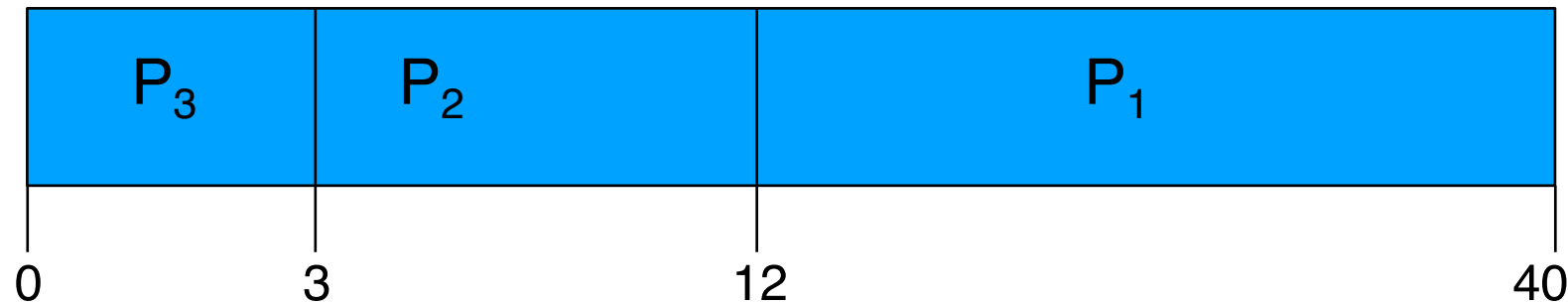
👤 平均等待时间是: $(0 + 28 + 37)/3 = 22$

👤 平均作业周转时间是: $(28 + 37 + 40)/3 = 35$



先来先服务(续)

💡 如果换一种执行顺序的话: P3 , P2 , P1



💡 等待时间分别是: $P1 = 12$; $P2 = 3$; $P3 = 0$

💡 平均等待时间是: $(12 + 3 + 0)/3 = 5$

💡 平均周转时间是: $(3 + 12 + 40)/3 = 18$

💡 第二种排列方式比第一种要好, 平均周转时间缩短为18

💡 FCFS的优缺点

💡 简单易行(+)

💡 如果短作业处在长作业的后面将导致等待时间变长。想像在超市里只买了一瓶矿泉水的你站在队伍的最后一位, 有何感想? (-)

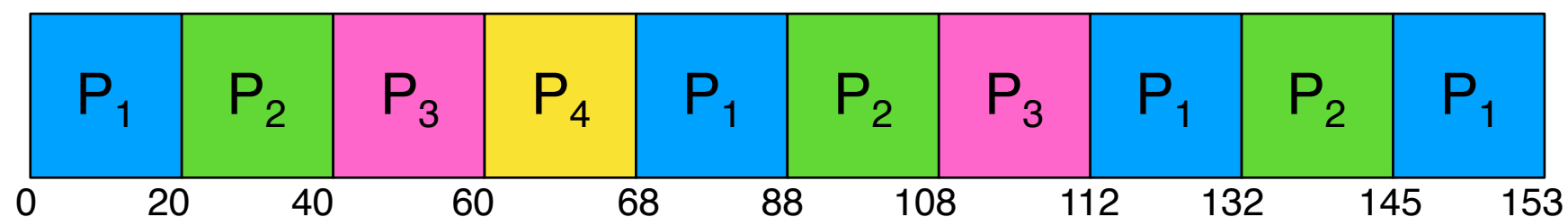
时间片轮转(ROUND ROBIN)



- 每个进程都可以得到相同的CPU时间(CPU时间片, **time slice**)，当时间片到达，进程将被剥夺CPU并加入就绪队列的尾部。
- 抢占式**调度算法
- n 个就绪队列中的进程和时间片 $q \Rightarrow$
 - 每个进程获得 $1/n$ 的CPU时间，大约是 q 个时间单位
 - 没有进程等待时间会超过 $(n-1)q$

RR例题(时间片=20)

Process	CPU Time
P1	68
P2	53
P3	24
P4	8



等待时间分别是：

$$P1=(68-20)+(112-88)+(145-132)=85$$

$$P2=(20-0)+(88-40)+(132-108)=92$$

$$P3=(40-0)+(108-60)=88$$

$$P4=(60-0)=60$$

平均等待时间 = $(85+92+88+60)/4=81.25$

平均周转时间 = $(153+145+112+68)/4=119.5$

如果采用FCFS算法，平均等待时间和平均周转时间分别是多少？

TAKE A BREAK



RR算法分析

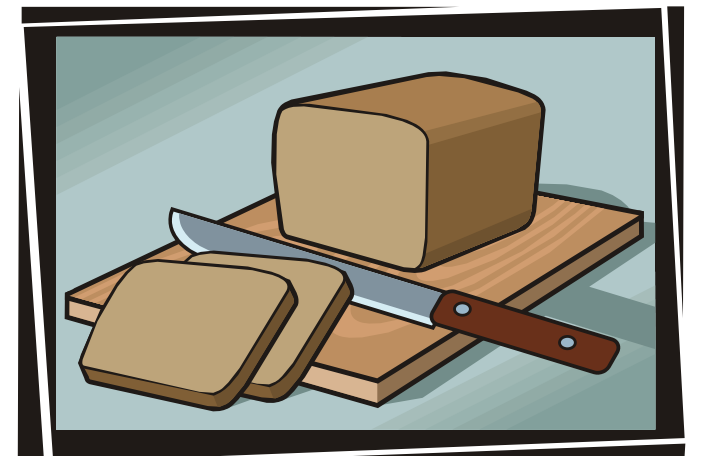
🧠 时间片（time slice）选取

- 🧠 取值太小：进程切换开销显著增大(不能小于进程切换的时间)
- 🧠 取值较大：响应速度下降（取值无穷大将退化成FCFS）
- 🧠 一般时间片的选取范围为 10ms~100ms
- 🧠 上下文切换的时间大概为 0.1ms~1ms（1%的CPU时间开销）

🧠 RR算法优缺点

- 🧠 公平算法(+)
- 🧠 对长作业带来额外的切换开销(-)

🧠 RR优于FCFS吗？



比较FCFS和RR

💡 10个进程，每个花费100个CPU时间

💡 假设RR 时间片为 1

💡 所有进程同时就绪队列之中

💡 结束时间：

进程 #	FCFS	RR
1	100	991
2	200	992
...
9	900	999
10	1000	1000

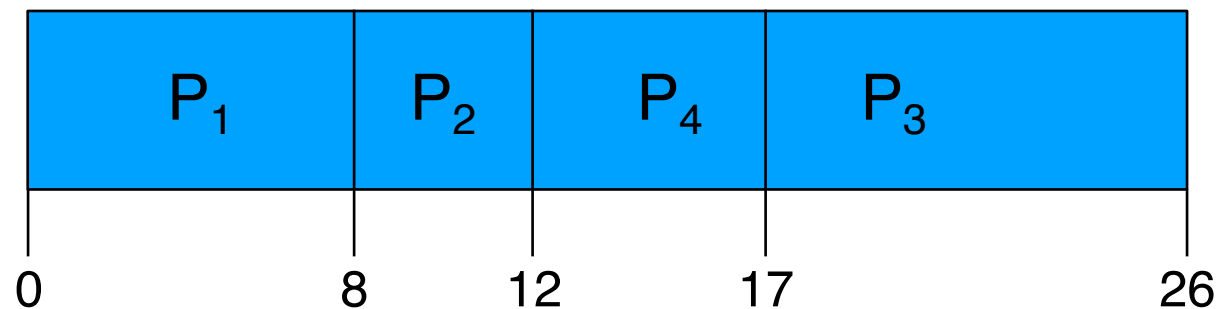
💡 RR和FCFS在同一时刻全部结束

💡 RR的平均周转时间相当糟糕！

最短作业优先(SJF)

💡 SJF(Shortest Job First): 下一次调度总是选择所需要CPU时间最短的那个作业（进程）。

	到达系统时间	所需CPU时间
P1	0	8
P2	1	4
P3	2	9
P4	3	5



💡 这是一个非抢占式算法，也可以改成抢占式SRTF。

SJF/SRTF算法分析

- 该算法总是将短进程移到长进程之前执行，因此平均等待时间最小，该算法被证明是最优的。

- 饥饿现象：长进程可能长时间无法获得CPU

- 预测技术

- 该算法需要事先知道进程所需的CPU时间

- 预测一个进程的CPU时间并非易事

- 优缺点：

- 优化了响应时间(+)

- 难以预测作业CPU时间(-)

- 不公平算法(-)



优先级调度(PRIORITY)

🧠 优先级通常为固定区间的数字，如[0, 10]:

🧠 数字大小与优先级高低的关系在不同系统中实现不一样，以Linux为例，0为最高优先级。

🧠 调度策略：下一次调度总是选择优先级最高的进程。

🧠 SJF是优先级调度的一个特例。

🧠 优先级调度可以是抢占式，也可以是非抢占式。



	执行时间	优先级
P1	10	8
P2	7	4
P3	1	9
P4	3	5
(所有进程0时刻同时到达)		

优先级的定义

静态优先级

-  优先级保持不变，但会出现不公平(饥饿)现象

动态优先级（退化Aging）

-  根据进程占用CPU时间：当进程占有CPU时间愈长，则慢慢降低它的优先级；
-  根据进程等待CPU时间：当进程在就绪队列中等待时间愈长，则慢慢提升它的优先级。

|Lecture 6

The End



下期预告

🧠 下次直播时间：2月26日 上午9:30

🧠 课程内容

🧠 番外篇 3 线程调度（brain burning）

🧠 答疑