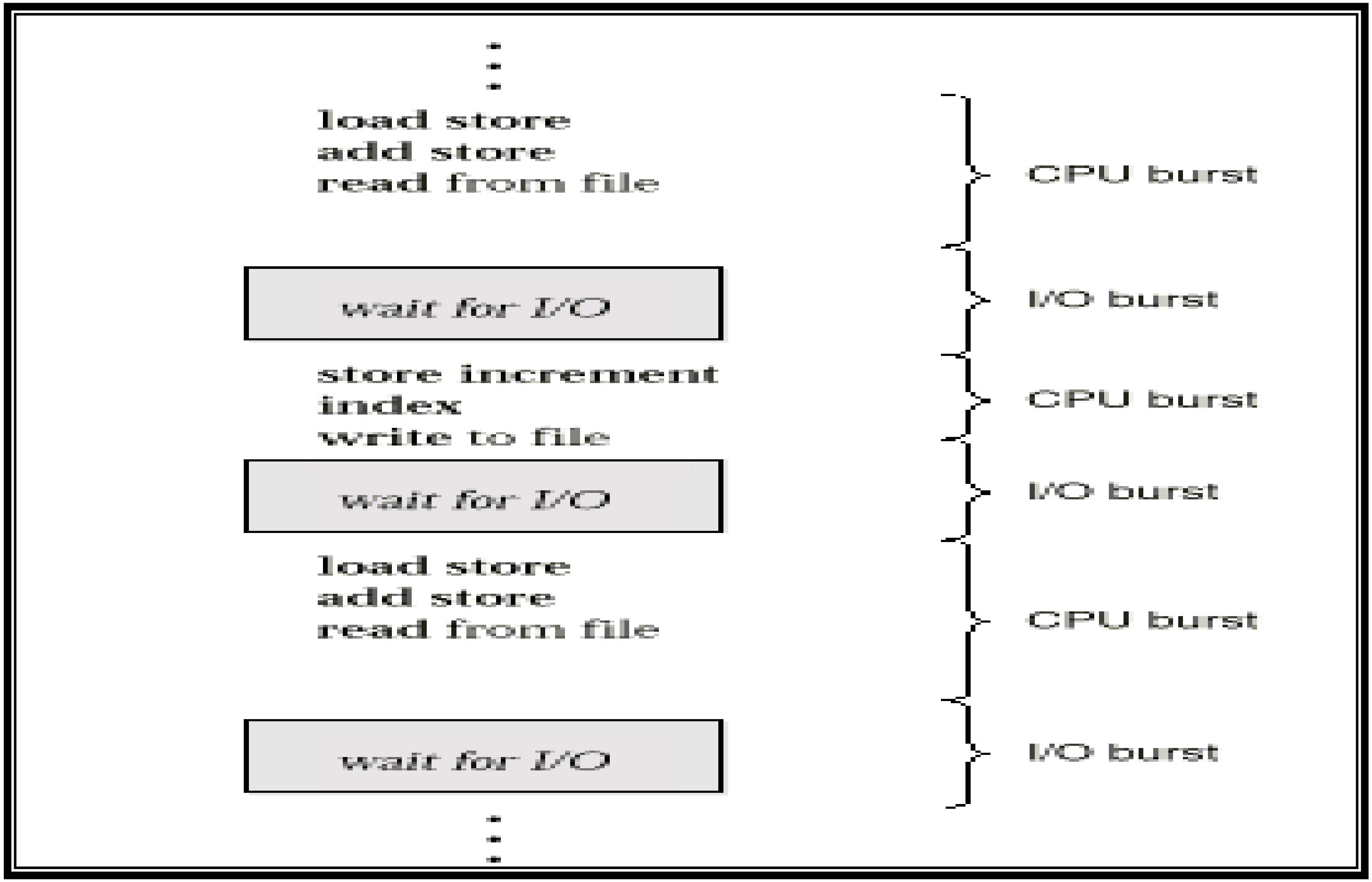
Chapter 5: CPU Scheduling

CPU Scheduling CPU调度

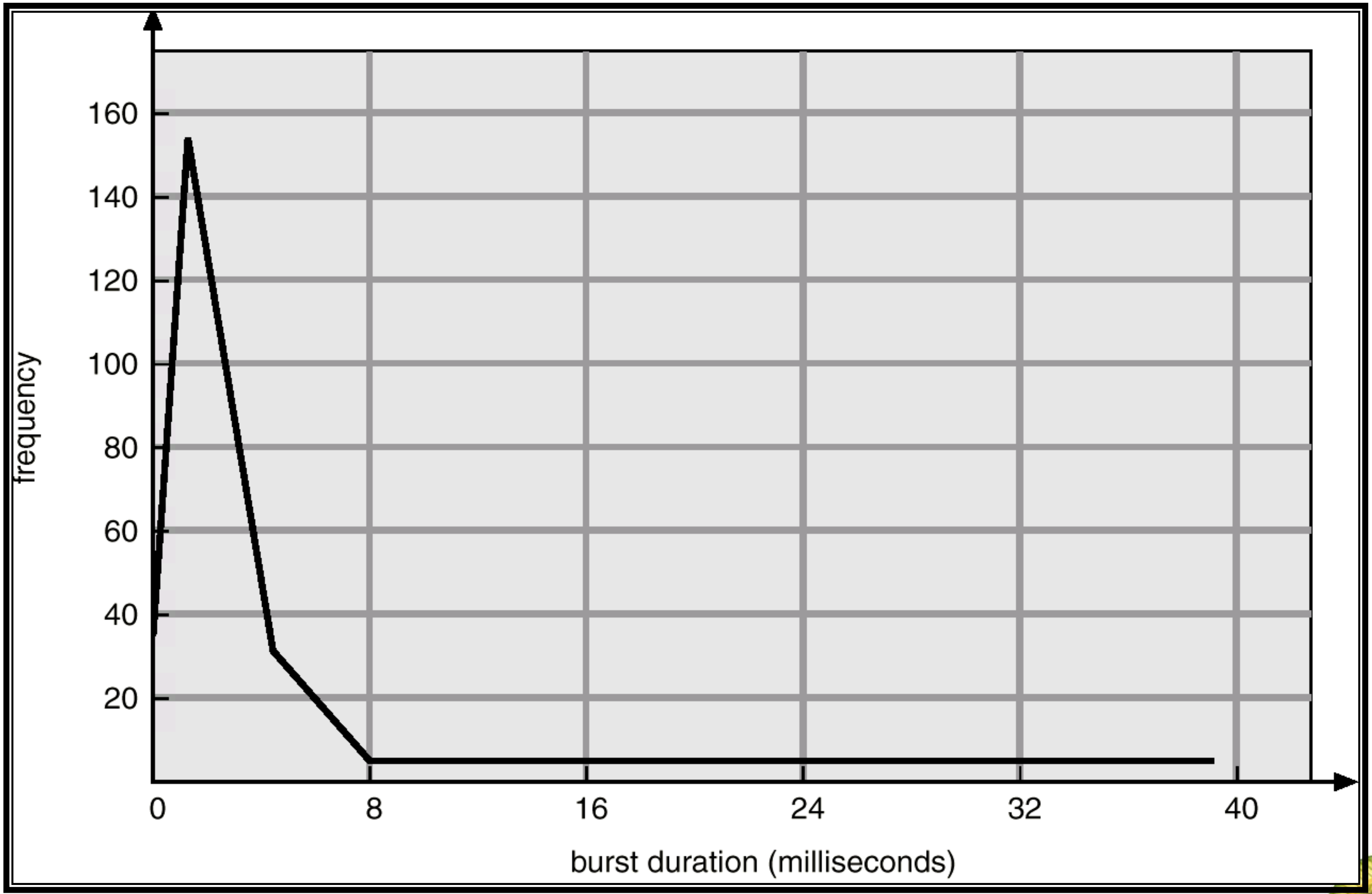
5.1 Basic Concepts

- CPU调度==处理机调度==进程调度
- Maximum CPU utilization obtained with multiprogramming
- CPU、I/O Burst Cycle脉冲（运行）周期– Process execution consists of a *cycle* of CPU execution and I/O wait.
- CPU Burst Time 、I/O Burst Time
- CPU-bound 、I/O-bound program （CPU型、I/O型程序）
- CPU burst distribution

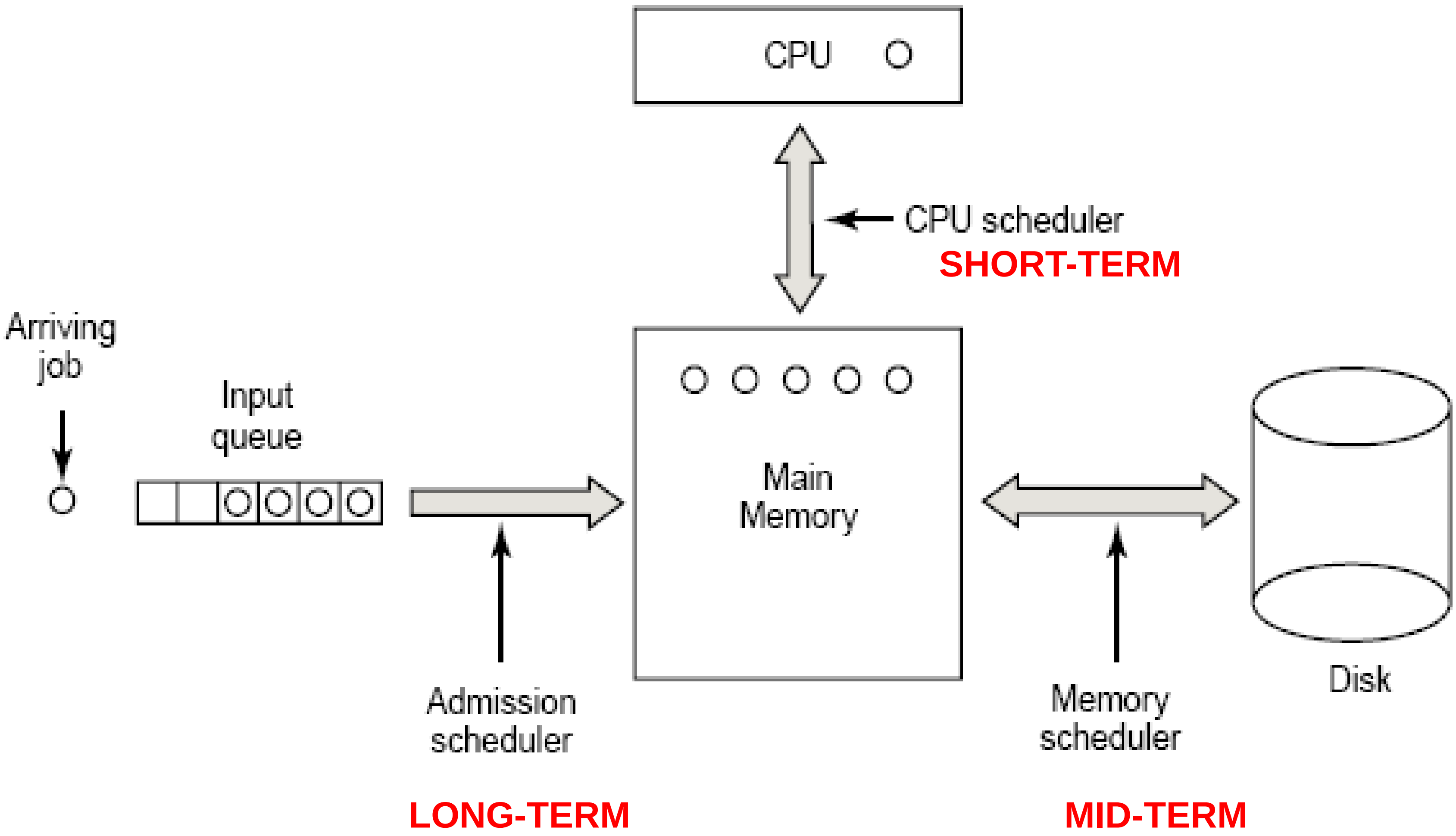
Fig 5.1 Alternating Sequence of CPU And I/O Bursts



Histogram of CPU-burst Times

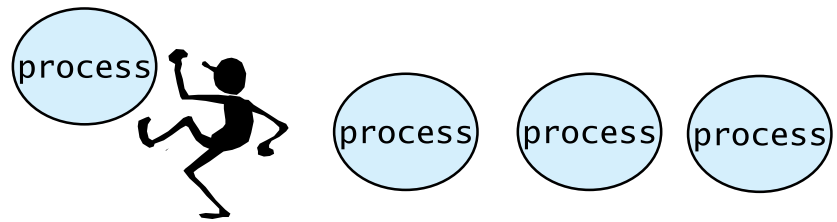


Three-level Scheduling



CPU Scheduler

- CPU scheduling decisions may take place when a process（调度的时机）：
  1. Switches from running to waiting state.
  2. Switches from running to ready state.
  3. Switches from waiting to ready.
  4. Terminates.



- 调度方式:
  - 非抢占式（Nonpreemptive）调度:调度程序一旦把处理机分配给某进程后便让它一直运行下去，直到进程完成或发生某事件而阻塞时，才把处理机分配给另一个进程。如上1、4
  - 抢占式（Preemptive）调度:当一个进程正在运行时，系统可以基于某种原则，剥夺已分配给它的处理机，将之分配给其它进程。剥夺原则有：优先权原则、短进程优先原则、时间片原则。如上2、3

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- Dispatch latency – time it takes for the dispatcher to stop one process and start another running

5.2调度算法的选择准则和评价

1.面向用户(User-oriented)的准则和评价

- **周转时间**Turnaround time： 进程从提交到完成所经历的时间。包括：在CPU上执行，就绪队列和阻塞队列中等待。
  - **周转时间 T=完成时间-提交时间**
  - **平均周转时间=∑周转时间/进程数**
  - **带权周转时间W= T(周转时间)/t(CPU执行时间)**
  - **平均带权周转时间=∑W/进程数**
- **响应时间**Response time：从进程提出请求到首次被响应（而不是输出结果）的时间段（在分时系统环境下）
- **等待时间**Waiting time – 进程在就绪队列中等待的时间总和
- **截止时间**：开始截止时间和完成截止时间——实时系统，与周转时间有些相似。
- 公平性：不因作业或进程本身的特性而使上述指标过分恶化。如长进程等待很长时间。
- 优先级：可以使关键任务达到更好的指标

调度算法的选择准则和评价

2. 面向系统的调度性能准则

- **吞吐量**Throughput：单位时间内所完成的进程数，跟进程本身特性和调度算法都有关系——批处理系统
  - 平均周转时间不是吞吐量的倒数，因为并发执行的进程在时间上可以重叠。如：在2小时内完成4个进程，而每个周转时间是1小时，则吞吐量是2个进程/小时
- **处理机利用率**CPU utilization：使CPU尽可能的忙碌
- **各种设备的均衡利用**：如CPU繁忙的进程和I/O繁忙的进程搭配——大中型主机

3. 调度算法本身的调度性能准则

- 易于实现
- 执行开销比较小

4. Optimization Criteria最优准则

- 最大的CPU利用率 Max CPU utilization
- 最大的吞吐量 Max throughput
- 最短的周转时间 Min turnaround time
- 最短的等待时间 Min waiting time
- 最短的响应时间 Min response time

- 公平

5.3 Scheduling Algorithms

5.3.1 First-Come, First-Served (FCFS) Scheduling先来先服务调度

5.3.2 Shortest-Job-First (SJF) Scheduling 短作业优先调度

5.3.3 Priority Scheduling 优先权调度

5.3.4 Round Robin (RR) 时间片轮转调度

5.3.5 Multilevel Queue Scheduling 多级队列调度

5.3.6 Multilevel Feedback Queue Scheduling多级反馈队列调度

- 高响应比优先调度算法 **Highest Response Ratio Next(HRRN)**
  - **响应比R = (等待时间 + 要求执行时间) / 要求执行时间**

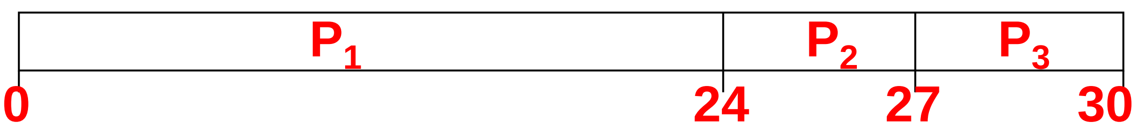
5.3.1 First-Come, First-Served (FCFS) Scheduling

- **FCFS算法**
  - **按照进程或作业提交顺序形成就绪状态的先后次序，分派CPU**
  - 当前进程或作业占用CPU，直到执行完或阻塞，才出让CPU（非抢占方式）
  - 在进程或作业唤醒后（如I/O完成），并不立即恢复执行，通常等到当前作业或进程出让CPU
  - 最简单的算法
- FCFS的特点
  - 比较有利于长进程，而不利于短进程。
  - 有利于CPU Bound的进程，而不利于I/O Bound的进程。

FCFS Scheduling(Cont.)

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
- The Gantt Chart for the schedule is:

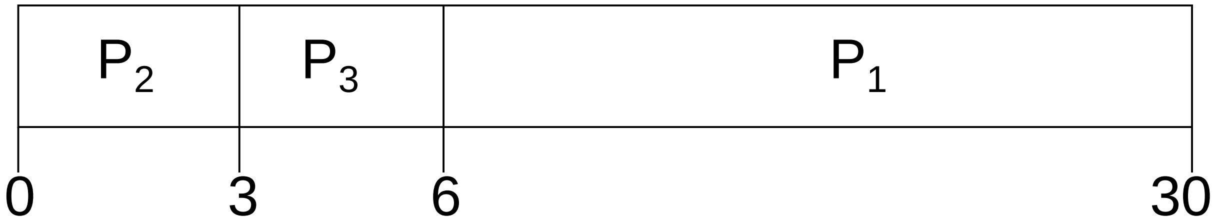


- Waiting time for $P1 = 0$; $P2 = 24$; $P3 = 27$
- Turnaround time for $P1 = 24$; $P2 = 27$; $P3 = 30$
- Average waiting time: $(0 + 24 + 27)/3 = 17$
- Average Turnaround time: $(24 + 27 + 30)/3 = 27$

turnaround time = termination time - arrival time.

FCFS Scheduling (Cont.)

- Suppose that the processes arrive in the order P_2 , P_3 , P_1 .
- The Gantt chart for the schedule is:



Process	CPU
P_1	24
P_2	3
P_3	3

  - *Waiting time : for P1 = 6; P2 = 0; P3 = 3*
  - *Turnaround time： for P1 = 30; P2 = 3; P3 = 6*
  - *Average waiting time : (6 + 0 + 3)/3 = 3*
  - *Average Turnaround time: (30 + 3 + 6)/3 = 13*
- Much better than previous case.
- *Convoy effect* short process behind long process

What if the arrival order is changed?

Waiting time = turnaround time - burst time.

5.3.2 Shortest-Job-First (SJF) Scheduling

- 又称为“**短进程优先**”SPF(Shortest Process First)；这是对FCFS算法的改进，其目标是减少平均周转时间。
- **SJF算法:**
  - **对预计执行时间短的作业（进程）优先分派处理机。**
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF)
- SJF is optimal – gives minimum average waiting time for a given set of processes.

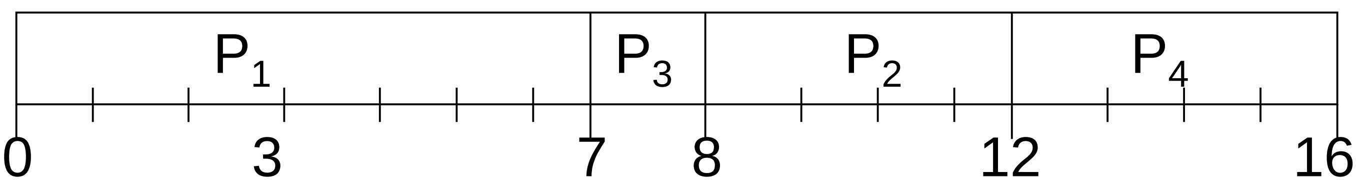
SJF的变型

- **最短剩余时间优先**SRT(Shortest Remaining Time)-基于抢占的SJF算法
  - 允许比当前进程剩余时间更短的进程来抢占
- **最高响应比优先**HRRN(Highest Response Ratio Next)
  - **响应比R = (等待时间 + 要求执行时间) / 要求执行时间**
  - 是FCFS和SJF的折衷

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- SJF (non-preemptive)



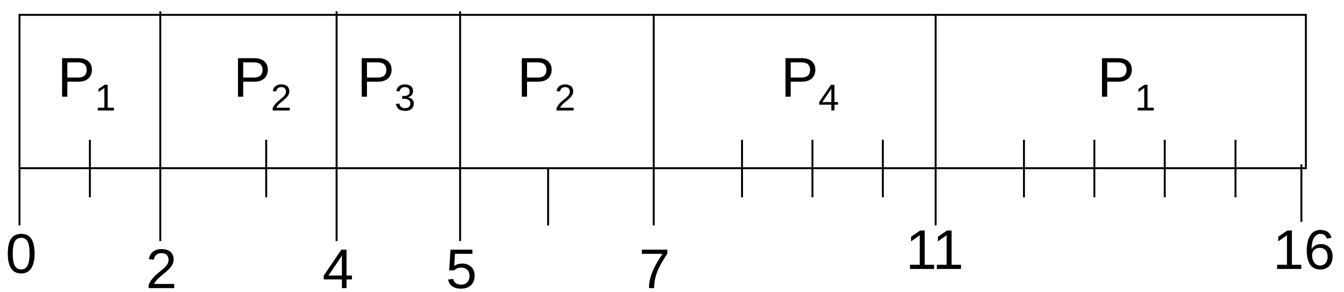
- *Average Turnaround time*=(7+10+4+11)/4=8
- *Average waiting time* = (0 + 6 + 3 + 7)/4 = 4

turnaround time = termination time - arrival time.

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- SJF (preemptive)



- *Average Turnaround time*=(16+5+1+6)/4=7
- *Average waiting time* = (9 + 1 + 0 +2)/4 = 3

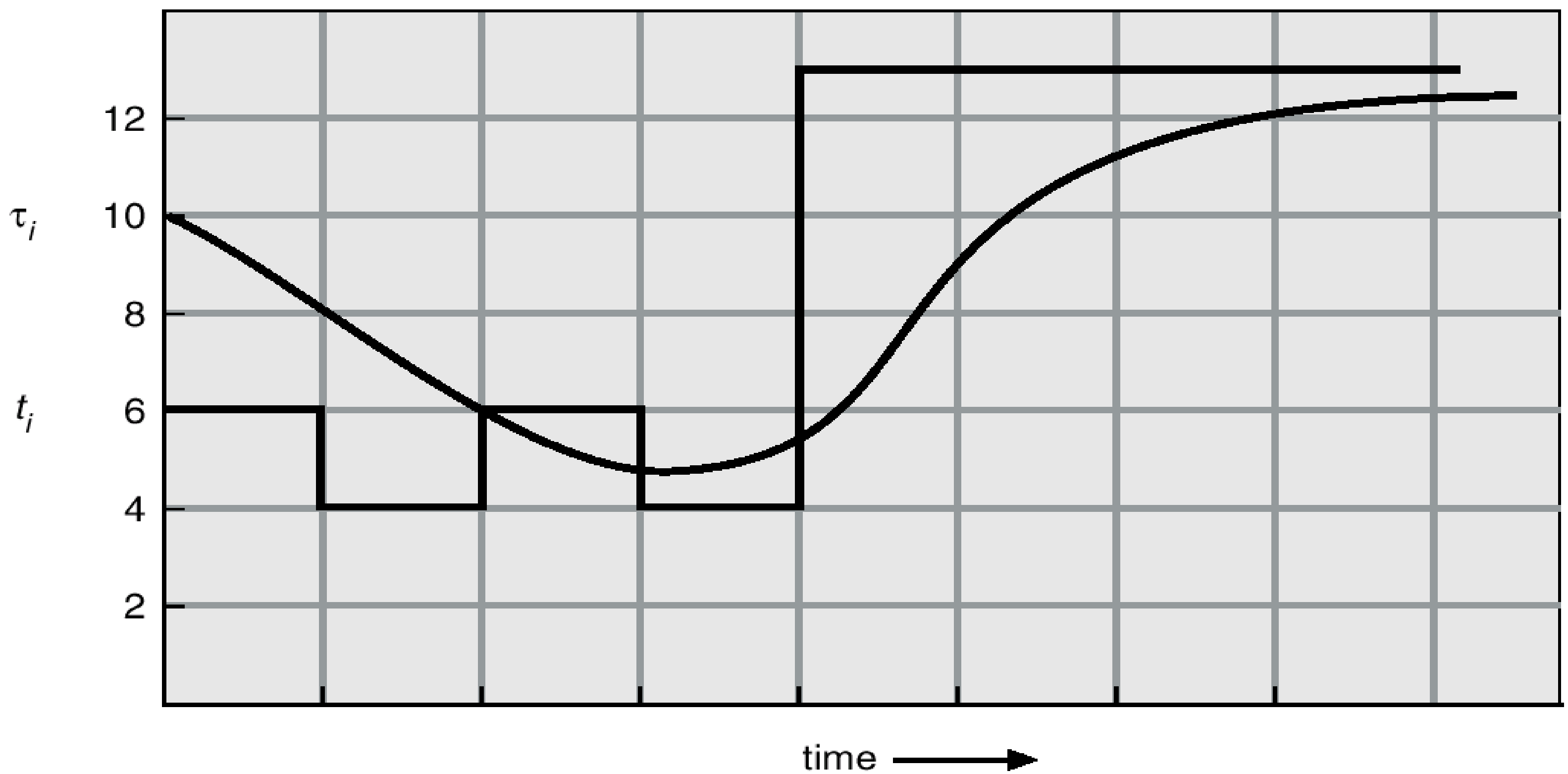
5.Determining Length of Next CPU Burst

- Can only estimate the length.
- Can be done by using the length of previous CPU bursts, using exponential averaging.

$$1.\ t_n = \text{actual lenght of } n^{th} \text{CPU burst}$$
$$2.\ \tau_{n+1} = \text{predicted value for the next CPU burst}$$
$$3.\ \alpha, 0 \le \alpha \le 1$$
$$4.\ \text{Define :}$$

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n .$$

Fig 5.3 Prediction of the Length of the Next CPU Burst



CPU burst (t_i)		6	4	6	4	13	13	13	. . .
"guess" (τ_i)	10	8	6	6	5	9	11	12	. . .

Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count.
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Only the actual last CPU burst counts.
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \alpha\, t_n -1 + \ldots$$

$$+ (1 - \alpha\,)^j\, \alpha\, t_n -1 + \ldots$$

$$+ (1 - \alpha\,)^{n=1}\, t_n\, \tau_0$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.

5.3.3 Priority Scheduling

- Associate a priority number with each process
- **该算法总是把处理机分配给就绪队列中具有最高优先权的进程。**常用以下两种方法来确定进程的优先权：
  - **静态优先权**:静态优先权是在创建进程时确定的，在整个运行期间不再改变。依据有：进程类型、进程对资源的要求、用户要求的优先权。
  - **动态优先权**:动态优先权是基于某种原则，使进程的优先权随时间改变而改变。
- 假定：**最小的整数 ≡ 最高的优先级**.

typically, a lower priority number indicates a higher priority

- SJF是以下一次CPU脉冲长度作为优先数的优先级调度

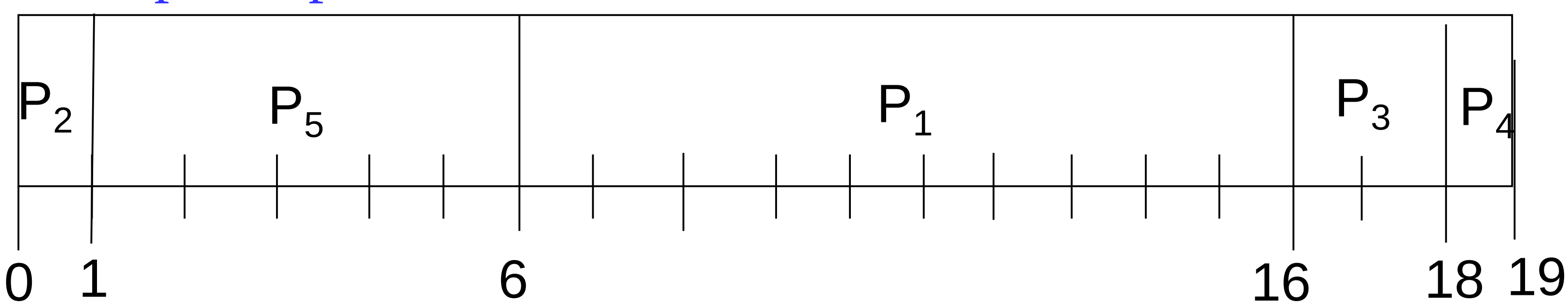
Priority Scheduling

- Is priority scheduling preemptive or non-reemptive?
  - Non-preemptive priority scheduling places higher-priority processes at the head of the queue
  - Preemptive priority scheduling requires a running process to be interrupted and preempted upon the arrival of a higher-priority process

Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- non-preemptive

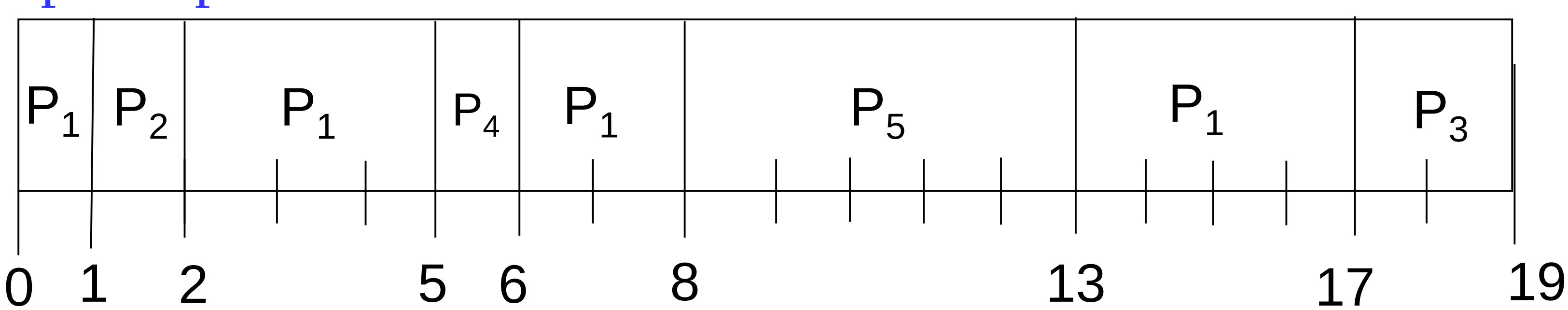


- *Average Turnaround time*=(16+1+18+19+6)/5=12
- *Average waiting time* = (6+0+16 +18 +1)/5 = 8.2

Example of Priority Scheduling

Process	Arrival Time	Burst Time	Priority
P_1	0	10	3
P_2	1	1	1
P_3	4	2	4
P_4	5	1	2
P_5	8	5	2

- preemptive

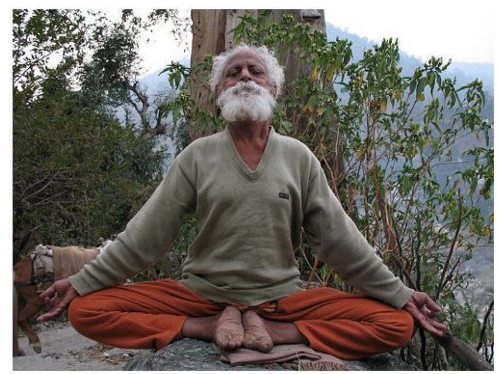


- *Average Turnaround time*=(17+1+15+1+5)/5=7.8
- *Average waiting time* = (7+0+13 +0 +0)/5 = 4

Priority Scheduling (Cont.)

- Problem ： Starvation(饥饿)– low priority processes may never execute.

(Rumor has it that ,When they shut down the IBM 7094 at MIT in 1973,they found a low-priority process that had been submitted in 1967 and had not yet been run)



- Solution ： Aging(老化)– as time progresses increase the priority of the process.——动态优先级

5.3.4时间片轮转调度Round Robin (RR)

- 基本思路：通过时间片轮转，提高进程并发性和响应时间特性，从而提高资源利用率。
- RR算法：
  - 将系统中所有的就绪进程按照FCFS原则，排成一个队列。
  - 每次调度时将CPU分派给队首进程，让其**执行一个时间片** (*time slice*) 。时间片的长度从几个ms到几百ms。
  - 在一个时间片结束时，发生时钟中断。
  - 调度程序据此暂停当前进程的执行，将其送到就绪队列的末尾，并通过上下文切换执行当前的队首进程。
  - 进程可以未使用完一个时间片，就出让CPU（如阻塞）。

5.3.4 Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.

RR情境

Round Robin

- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.

  响应时间=n*q

- Performance
  - q large $\Rightarrow$ FIFO

    Why?

  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

- 时间片长度的影响因素：
  - 就绪进程的数目：**数目越多，时间片越小**（当响应时间一定时）
  - 系统的处理能力：应当使用户输入通常在一个时间片内能处理完，否则使响应时间，平均周转时间和平均带权周转时间延长。

Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:

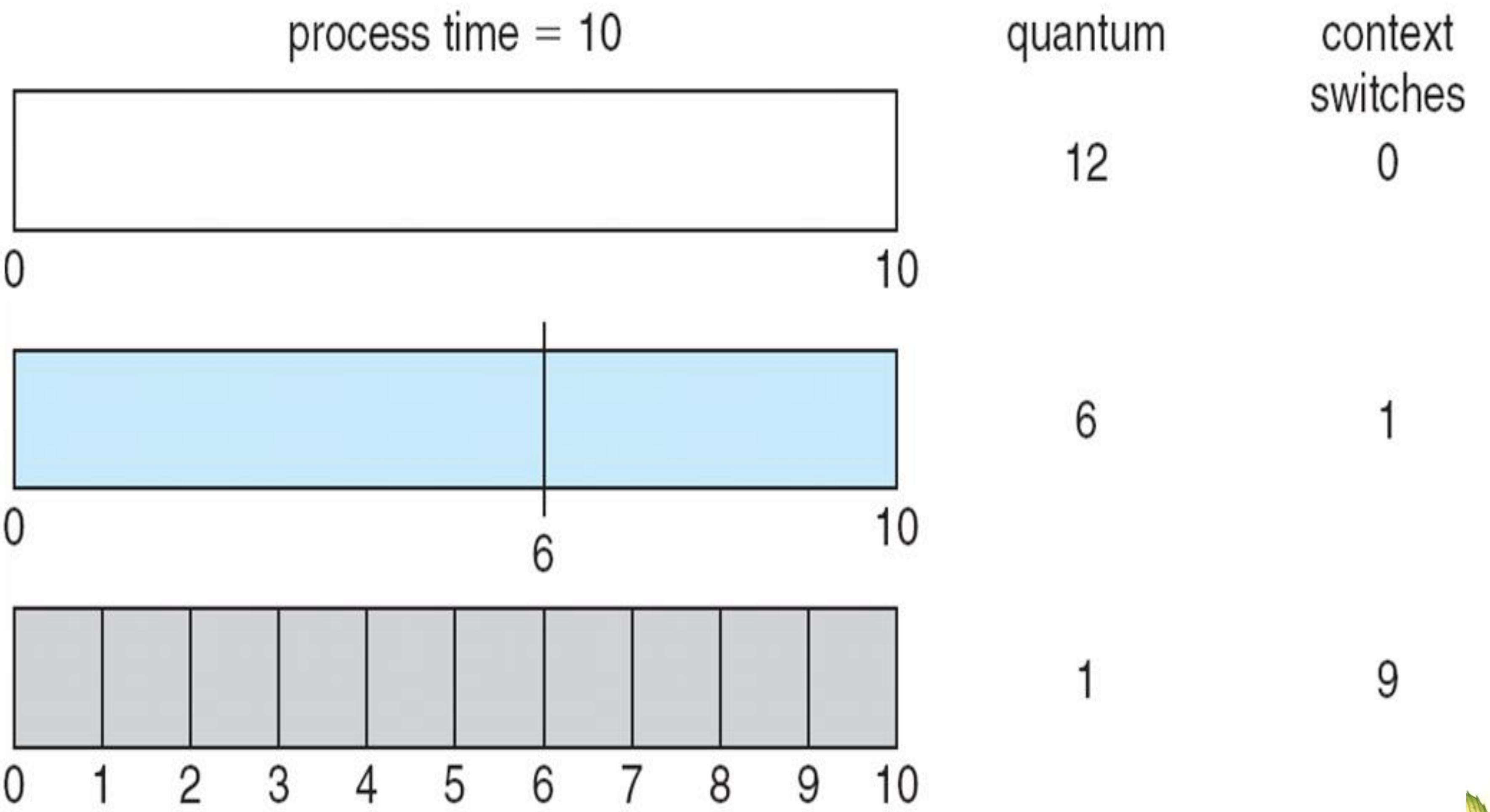
P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 37 57 77 97 117 121 134 154 162

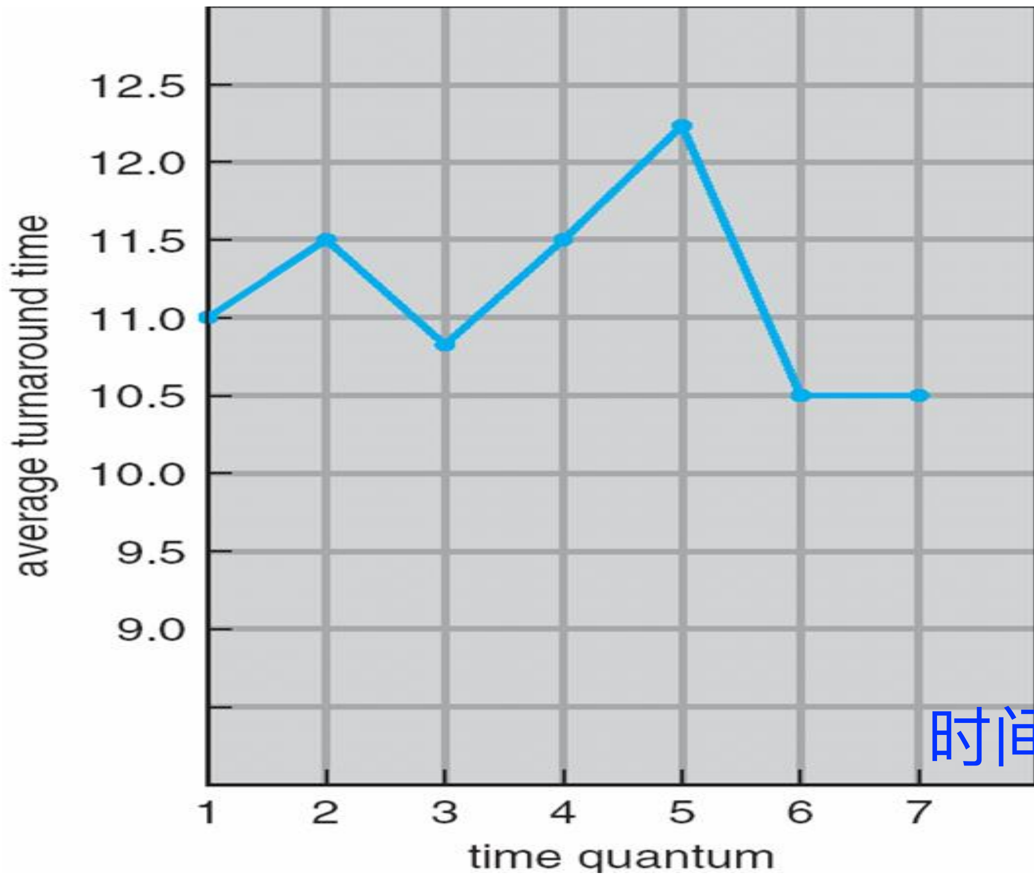
- ***Average Turnaround time=(134+37+162+121)/4=113.5***
- ***Average waiting time = (81+20+94 +97)/4= 70.5***

- Typically, higher average turnaround than SJF, but better *response*.

Fig 5.4 Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts should be shorter than q

时间片优先级都在一个队列里

5.3.5 多级队列调度

本算法引入多个就绪队列，通过各队列的区别对待，达到一个综合的调度目标

- 根据进程的性质或类型的不同，将**就绪队列再分为若干个子队列**。
- **每个作业固定归入一个队列**。
- 各队列的不同处理：**不同队列可有不同的优先级、时间片长度、调度策略等。如：系统进程、用户交互进程、批处理进程等。**

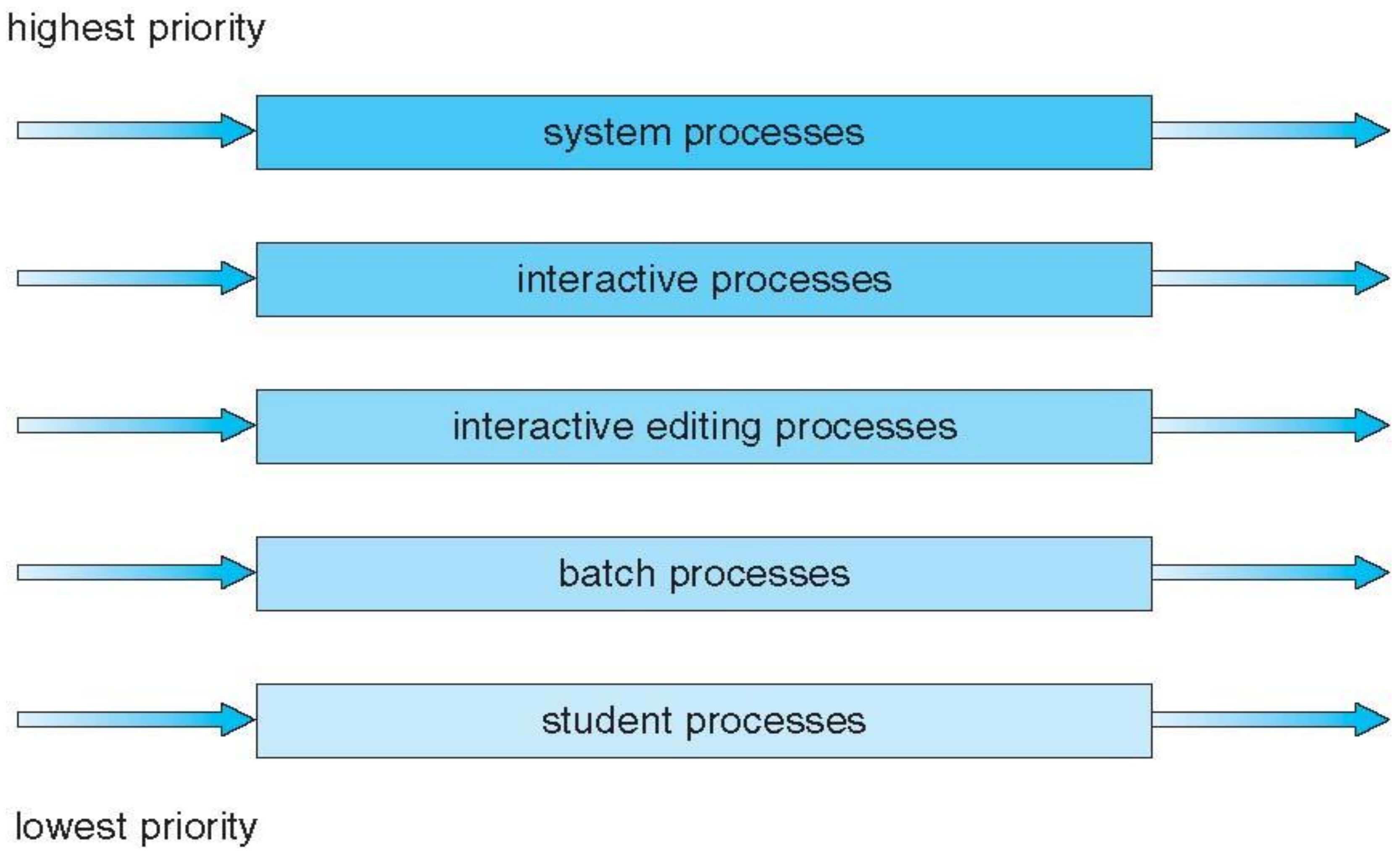
Multilevel Queue Scheduling

- Ready queue is partitioned into separate queues:
  - foreground (interactive) 前台（交互式）
  - background (batch) 后台 （批处理）
- Each queue has its own scheduling algorithm:
  - foreground – RR 前台轮转
  - background – FCFS

Multilevel Queue Scheduling (Cont.)

- 多级队列算法调度须在队列间进行
  - **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

Multilevel Queue Scheduling (Cont.)



5.3.6 Multilevel Feedback Queue Scheduling

优先级会变化 现在的操作系统

- **多级反馈队列**算法是时间片轮转算法和优先级算法的综合和发展。优点：
  - 为提高系统吞吐量和缩短平均周转时间而照顾短进程
  - 为获得较好的I/O设备利用率和缩短响应时间而照顾I/O型进程
  - 不必估计进程的执行时间，动态调节

多级反馈队列算法

- 设置多个就绪队列，分别赋予不同的优先级，如逐级降低，队列1的优先级最高。每个队列执行时间片的长度也不同，规定优先级越低则时间片越长，如逐级加倍
- 新进程进入内存后，先投入队列1的末尾，按FCFS算法调度；**若按队列1一个时间片未能执行完，则降低投入到队列2的末尾，同样按FCFS算法调度；**如此下去，降低到最后的队列，则按"时间片轮转"算法调度直到完成。
- 仅当较高优先级的队列为空，才调度较低优先级的队列中的进程执行。如果进程执行时有新进程进入较高优先级的队列，则抢占执行新进程，并把被抢占的进程投入原队列的末尾。

几点说明

- **I/O型进程**：让其进入最高优先级队列，以及时响应I/O交互。通常执行一个小时间片，要求可处理完一次I/O请求的数据，然后转入到阻塞队列。
- **计算型进程**：每次都执行完时间片，进入更低级队列。最终采用最大时间片来执行，减少调度次数。
- **I/O次数不多**，而主要是CPU处理的进程：在I/O完成后，放回优先I/O请求时离开的队列，以免每次都回到最高优先级队列后再逐次下降。
- 为适应一个进程在不同时间段的运行特点，I/O完成时，提高优先级；时间片用完时，降低优先级；

Example of Multilevel Feedback Queue

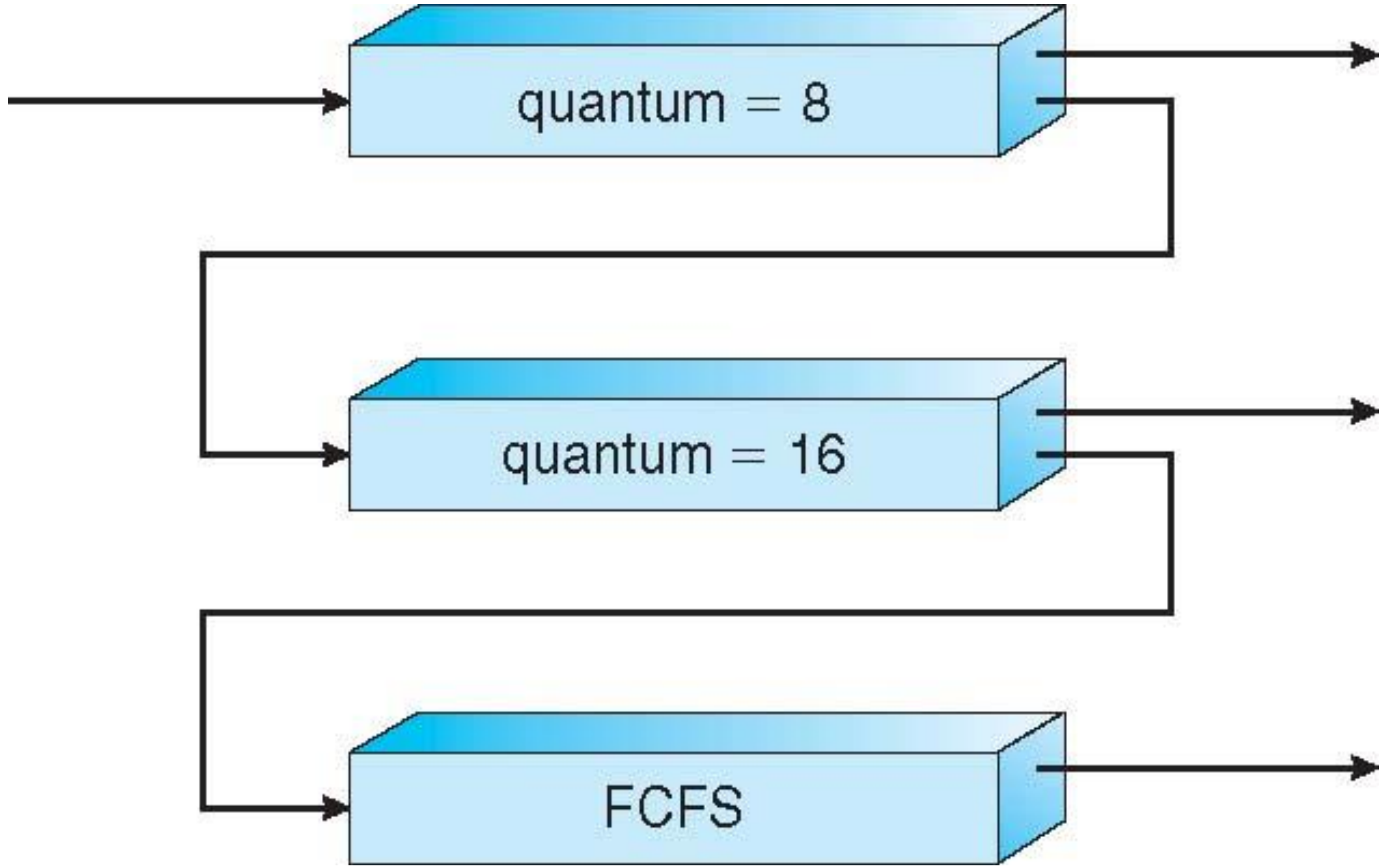
- Example of Multilevel Feedback Queue Three queues:
  - Q_0 – time quantum 8 milliseconds
  - Q_1 – time quantum 16 milliseconds
  - Q_2 – FCFS

Example of Multilevel Feedback Queue

Scheduling

- A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1.
- At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.

Fig 5.7 Multilevel Feedback Queues



进程调度例题分析

假定在一个处理机上执行以下五个进程：

进程号	到达时间	运行时间	优先数
P1	0	35	4
P2	10	30	3
P3	15	45	2
P4	20	20	5
P5	30	30	2

分别采用FCFS、SPF和非抢占优先数三种调度算法时：

①画出调度Gantt图；

②计算每个进程的周转时间 ；

③计算平均周转时间。

进程调度例题分析

FCFS：

P1	P2	P3	P4	P5

35 65 110 130 160

SPF（非抢占）：

P1	P4	P2	P5	P3

35 55 85 115

160

优先数（非抢占）：

P1	P4	P2	P3	P5

35 55 85 130

160

算法	每个进程周转时间					平均周转时间
	P1	**P2**	**P3**	**P4**	**P5**	
FCFS	**35**	**55**	**95**	**110**	**130**	**85**
SPF	**35**	**75**	**145**	**35**	**85**	**75**
优先数	**35**	**75**	**115**	**35**	**130**	**78**

5.4* Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- *Homogeneous processors* within a multiprocessor
- *Load sharing*
- *Asymmetric multiprocessing* – only one processor accesses the system data structures, alleviating the need for data sharing

5.5* Thread Scheduling

- Local Scheduling – How the threads library decides which thread to put onto an available LWP

- Global Scheduling – How the kernel decides which kernel thread to run next

Pthread Scheduling API

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[])
{
     int i;
    pthread t tid[NUM THREADS];
    pthread attr t attr;
    /* get the default attributes */
    pthread_attr_init(&attr);
    /* set the scheduling algorithm to PROCESS or SYSTEM */
    pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
    /* set the scheduling policy - FIFO, RT, or OTHER */
    pthread_attr_setschedpolicy(&attr, SCHED_OTHER);
    /* create the threads */
    for (i = 0; i < NUM_THREADS; i++)
        pthread_create(&tid[i],&attr,runner,NULL);
```

Pthread Scheduling API

```
    /* now join on each thread */
    for (i = 0; i < NUM THREADS; i++)
        pthread_join(tid[i], NULL);
}
 /* Each thread will begin control in this
    function */
void *runner(void *param)
{
    printf("I am a thread\n");
    pthread exit(0);
}
```

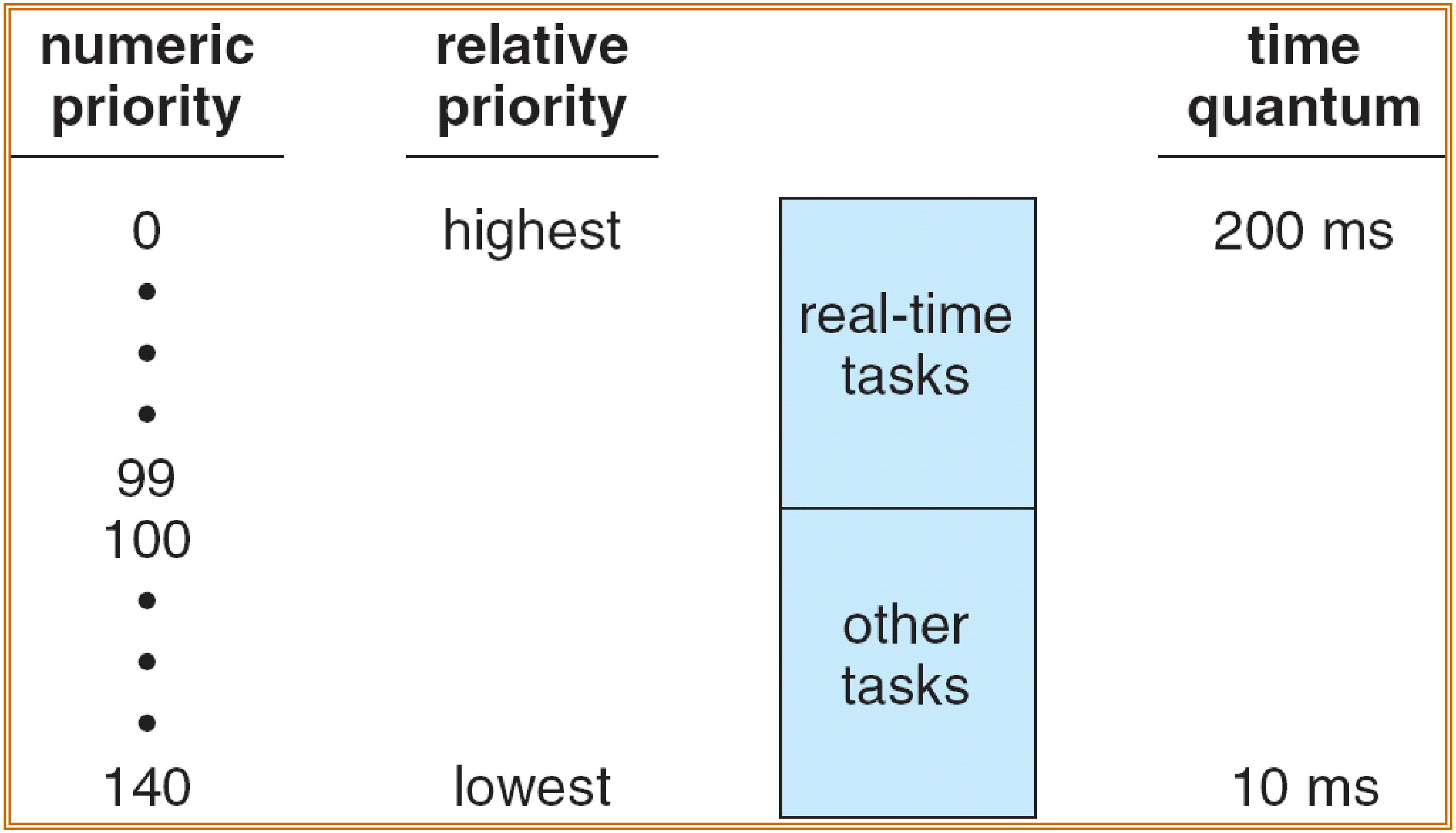
5.6 Operating System Examples

- Solaris scheduling
- Windows XP scheduling
- Linux scheduling

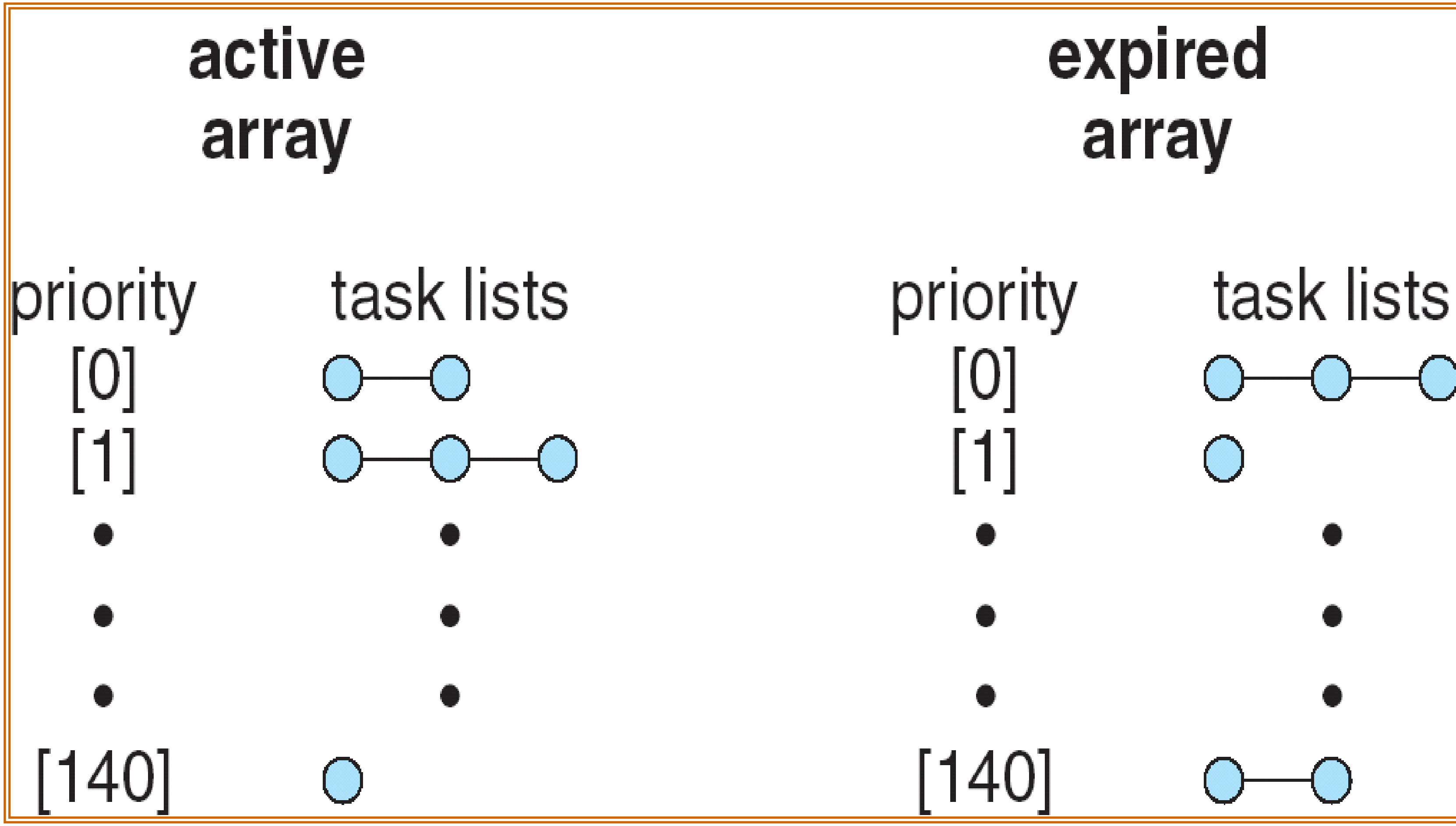
Linux Scheduling

- Two algorithms: time-sharing and real-time
- Time-sharing
  - Prioritized credit-based – process with most credits is scheduled next
  - Credit subtracted when timer interrupt occurs
  - When credit = 0, another process chosen
  - When all processes have credit = 0, recrediting occurs
    - Based on factors including priority and history
- Real-time
  - Soft real-time
  - Posix.1b compliant – two classes
    - FCFS and RR
    - Highest priority process always runs first

The Relationship Between Priorities and Time-slice length



List of Tasks Indexed According to Prorities



Summary

- CPU Burst Time 、I/O Burst Time
- CPU-bound program（CPU型程序）、I/O-bound program（I/O型程序）
- long-term scheduler（长程调度）、medium-term scheduler（中程调度）short-term scheduler（短程调度）
- time slicing（时间片）
- response time（响应时间）、turnaround time（周转时间）、waiting time（等待时间）、Average Turnaround time（平均周转时间）Average waiting time（平均等待时间）
- preemptive scheduling（抢占式调度）Nonpreemptive scheduling（非抢占式调度）
- throughput（吞吐量）

Summary

- selecting a process from the ready queue and allocating CPU to it
- scheduling algorithms
  - first-come, first served (FCFS)
  - shortest job first (SJF)
    - provably optimal, but difficult to know CPU burst
  - general priority scheduling
    - starvation, and aging
  - round-robin (RR)
    - for time-sharing, interactive system
    - problem: how to select the time quantum?
  - Multilevel queue
    - different algorithms for different classes of processes
  - Multilevel feedback queue
    - allow process to move from one (ready) queue to another

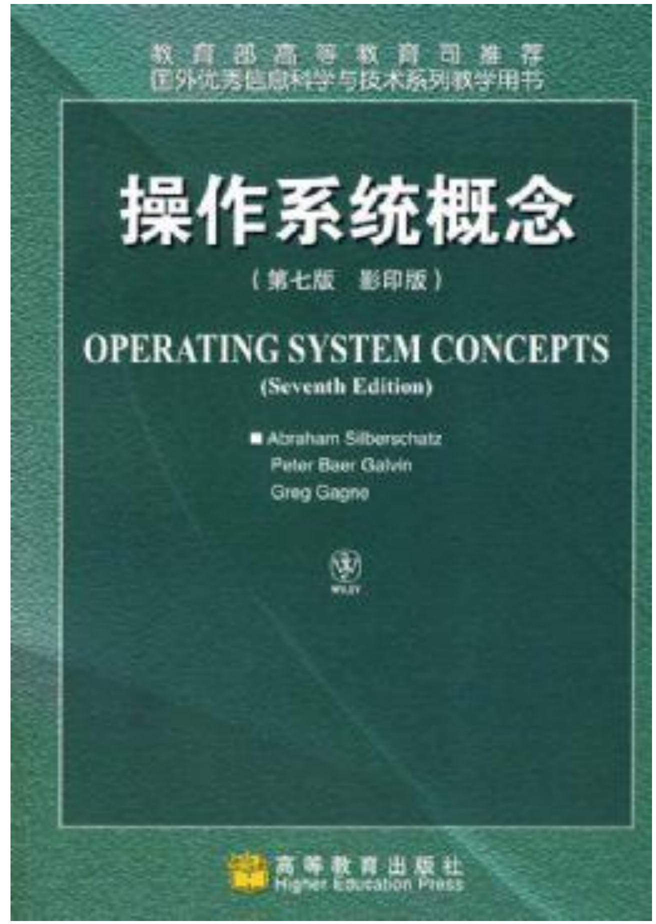
- 习题分析

homework

- 5.0、5.4、5.5、5.11

Reading Assignments

- Read for this week:
  - Chapters 5
    of the text book:
- Read for next week:
  - Chapters 6
    of the text book:



End of Chapter