操作系统原理及应用

李　伟

xchlw@seu.edu.cn

计算机科学与工程学院、软件学院
江苏省网络与信息安全重点实验室

Chapter 5 CPU Scheduling

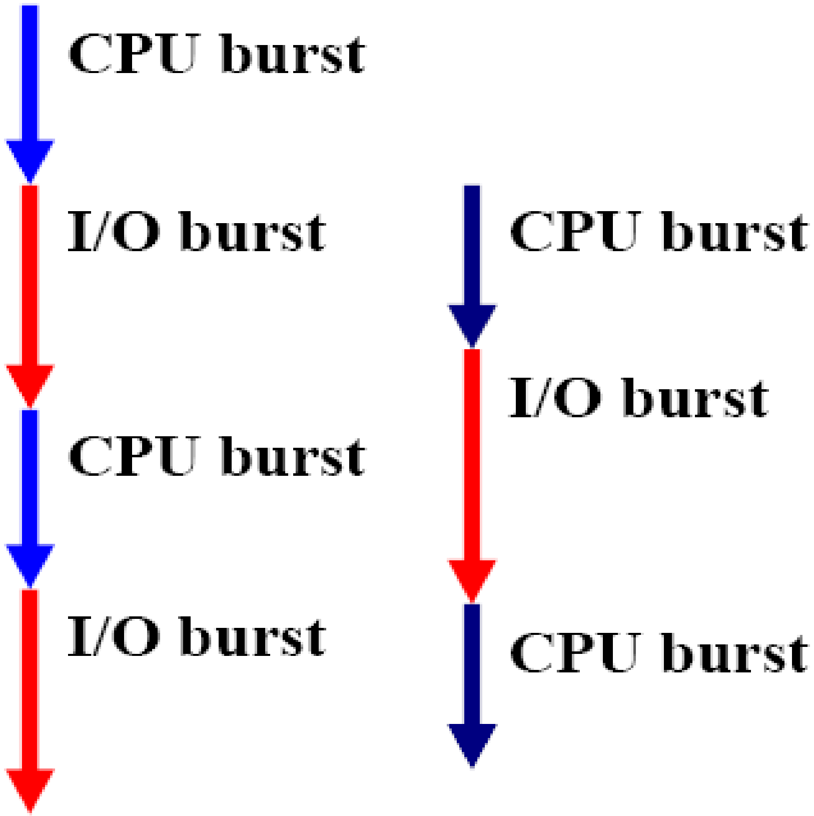
Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Thread Scheduling
- Algorithm Evaluation
- Operating System Examples

Basic Concepts

- **CPU scheduling is the basis of multiprogrammed operating systems**
- **Maximum CPU utilization obtained with multiprogramming**
- **The success of CPU scheduling depends on an property of processes——CPU–I/O Burst Cycle**
  - **Process execution consists of a *cycle* of CPU execution and I/O wait.**

CPU-I/O Burst Cycle



CPU burst

I/O burst

CPU burst

I/O burst

CPU burst

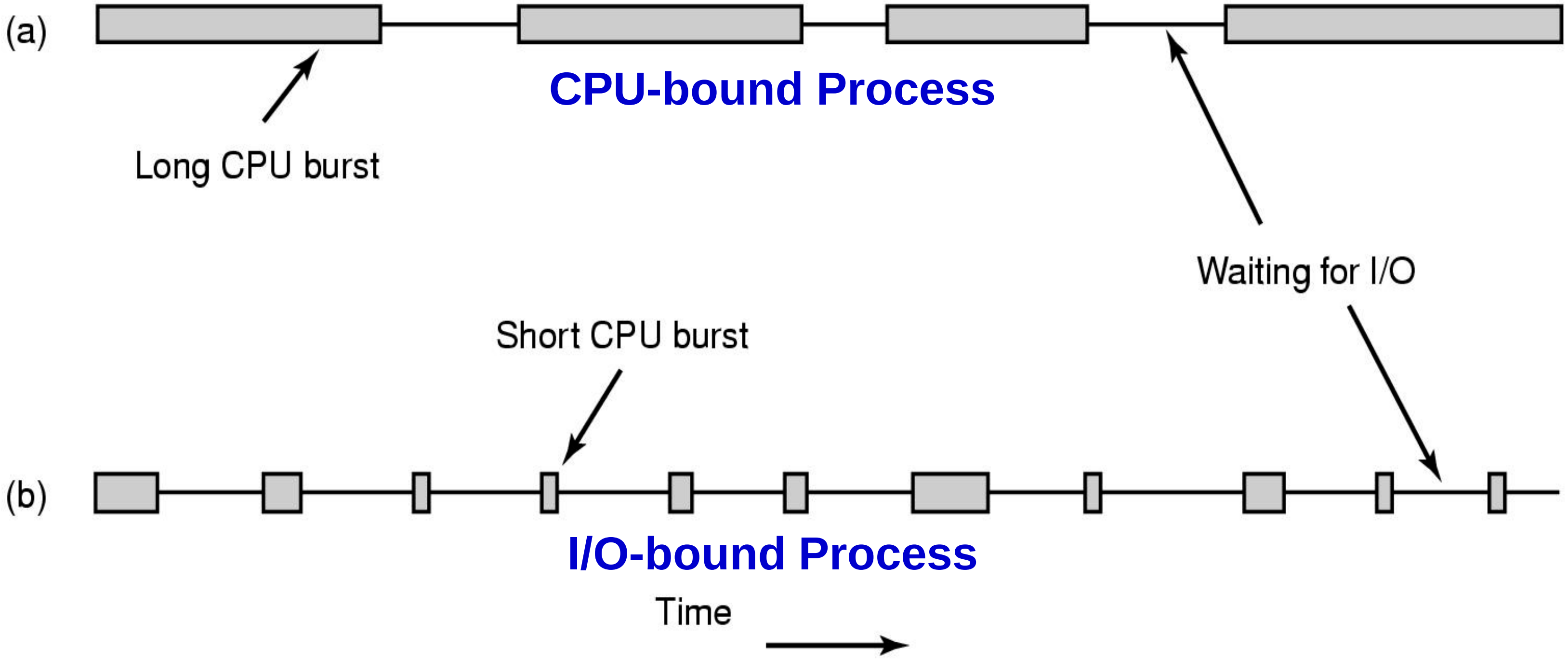
I/O burst

CPU burst

- Process execution repeats the CPU burst and I/O burst cycle.
- When a process begins an I/O burst, another process can use the CPU for its CPU burst.

CPU-bound and I/O-bound

- A process is CPU-bound if it generates I/O requests infrequently, using more of its time doing computation.
- A process is I/O-bound if it spends more of its time to do I/O than it spends doing computation.
- A CPU-bound process might have a few very long CPU bursts.
- An I/O-bound process typically has many short CPU bursts.



CPU-bound Process

I/O-bound Process

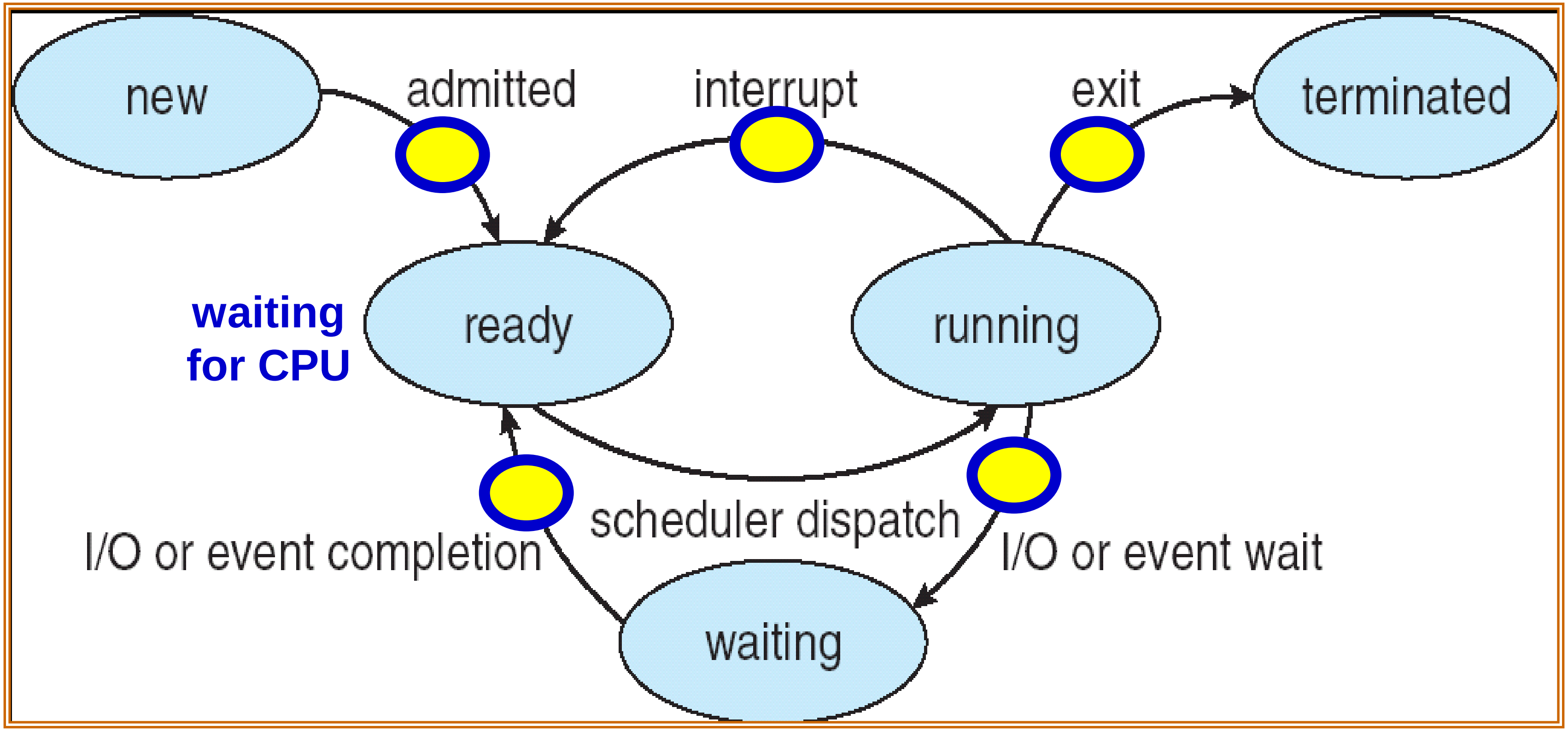
CPU Scheduler

- When the CPU is idle, the OS must select another ready process to run.
- This selection process is carried out by the ***short-term scheduler*** (or ***CPU scheduler***).
- The CPU scheduler selects a process from the ready queue, and allocates the CPU to it.
- There are many ways to organize the ready queue (e.g. FIFO).

Circumstances that scheduling may take place

converting to process

reclaim resource
destroy process



waiting
for CPU

waiting for I/O or event

Circumstances that scheduling may take place

- A process switches from the running state to the terminated state.
- A process switches from the running state to the wait state (*e.g*., doing for I/O).
- A process switches from the running state to the ready state (*e.g*., an interrupt occurs).
- A process switches from the wait state to the ready state (*e.g*., I/O completion).
- A process switches from the new state to the ready state (*e.g*., a higher priority process ready).

Preemptive vs. Non-preemptive

- Preemptive scheduling
  - scheduling occurs in case 3, case 4 and case 5.
  - Incuring a cost associated with access to shared data.
  - What happens if the kernel is in its critical section modifying some important data?
  - The kernel must pay special attention to this situation and, hence, is more complex.

Preemptive vs. Nonpreemptive

- Nonpreemptive scheduling
  - scheduling occurs when a process voluntarily terminates (case 1) or enters the wait state (case 2).
  - Once the CPU has been allocated to a process, the process keeps the CPU until it either terminates or switches to the waiting state.
  - Simple, but very inefficient

作业1

- 说明抢占式调度与非抢占式调度的区别。对于计算中心，上述两种调度方式哪一种比较适合？

Dispatcher

- **Dispatcher module gives control of the CPU to the process selected by the short-term scheduler.**
- **Its functions involve:**
  - **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.**

The dispatcher should be as fast as possible.

Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Thread Scheduling
- Algorithm Evaluation
- Operating System Examples

Scheduling Criteria

- There are many criteria for comparing different scheduling algorithms.
- Here are five common ones:
  - CPU utilization (CPU利用率)
  - Throughput (吞吐量)
  - Turnaround time (周转时间)
  - Waiting time (等待时间)
  - Response time (响应时间)

CPU Utilization

- We want to keep the CPU as busy as possible.
- CPU utilization ranges from 0 to 100 percent.
- Normally 40% or lower is lightly loaded and 90% or higher is heavily loaded.

Throughput

- The number of processes completed per time unit is called ***throughput***.
- Higher throughput means more jobs get done.
- However, for long processes, this rate may be one job per hour, and, for short jobs, this rate may be 10 per minute.

Turnaround Time

- The time period from job submission to completion is the turnaround time.
- From a user's point of view, turnaround time is more important than CPU utilization and throughput.
- Turnaround time is the sum of
  - waiting time before entering the system
  - waiting time in the ready queue
  - waiting time in all other events (e.g., I/O)
  - time the process actually running on the CPU

Waiting Time

- Waiting time is the sum of the periods that a process spends waiting in the ready queue.
- Why only ready queue?
  - CPU scheduling algorithms do not affect the amount of time during which a process executes or does I/O.
  - However, CPU scheduling algorithms do affect the time that a process stays in the ready queue

Response Time

- The time from the submission of a request (in an interactive system) to the first response is called response time. It does not include the time that it takes to output the response.
- For example, in front of your workstation, you perhaps care more about the time between hitting the Return key and getting your first output than the time from hitting the Return key to the completion of your program (e.g., turnaround time).

Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

In most cases, we optimize the average value.

However, under some cases, it is desirable to optimize the minimum or maximum values rather than the average.

Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Thread Scheduling
- Algorithm Evaluation
- Operating System Examples

Scheduling Algorithms

- First-Come-First-Served Scheduling (FCFS)
- Shortest-Job-First Scheduling (SJF)
- Priority Scheduling
- Round-Robin Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

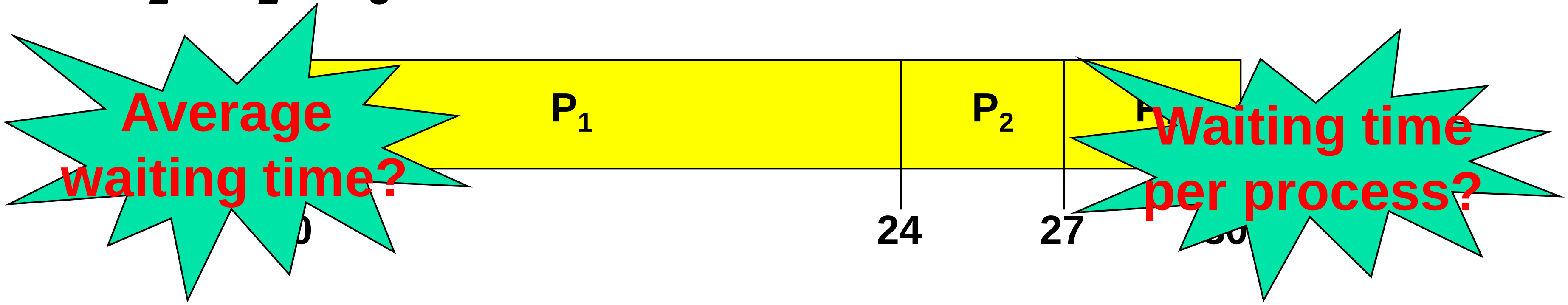
First-Come-First-Served (FCFS) Scheduling

- The process that requests the CPU first is allocated the CPU first.
- This can easily be implemented using a queue.
- FCFS is not preemptive. Once a process has the CPU, it will occupy the CPU until the process completes or voluntarily enters the wait state.

FCFS Scheduling

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:



Average waiting time?

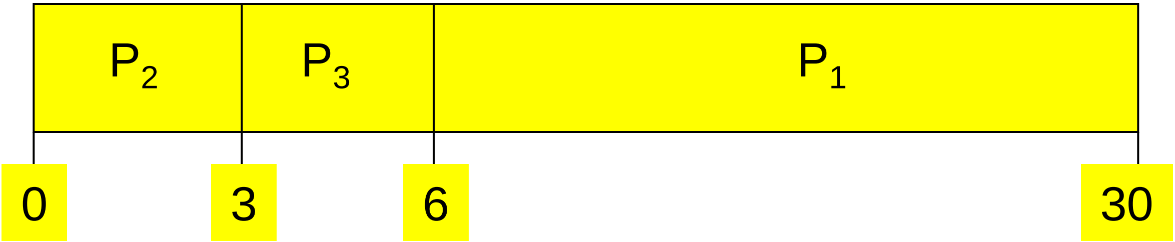
Waiting time per process?

FCFS Scheduling

Suppose that the processes arrive in the order

P_2 , P_3 , P_1 .

- **The Gantt chart for the schedule is:**



- **Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$**
- **Average waiting time: (6 + 0 + 3)/3 = 3**
- **Much better than previous case.**

FCFS Problems

- It is easy to have the ***convoy effect***: all the processes wait for the one big process to get off the CPU. **The utilization of CPU and Device may be low**. ***Why?***
- **It is in favor of long processes and may not be fair to those short ones. What if your 1-minute job is behind a 10-hour job?**
- **It is troublesome for time-sharing systems, where each user needs to get a share of the CPU at regular intervals.**

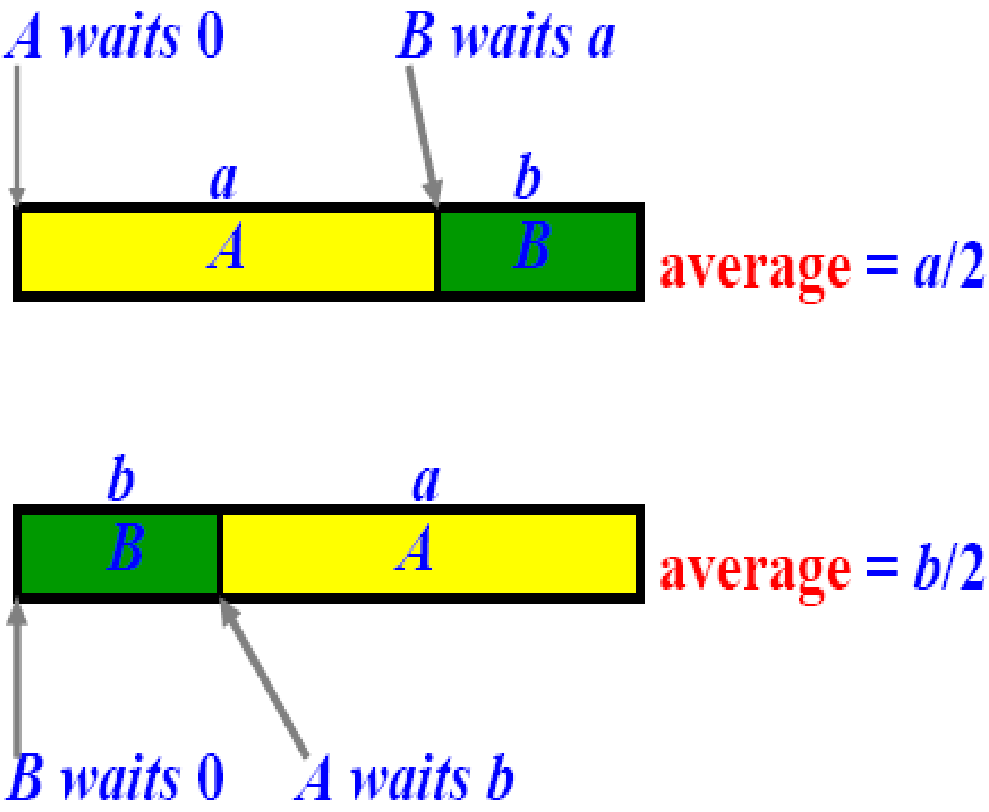
Shortest-Job-First (SJF) Scheduling

- **Associate each process with the length of its next CPU burst.**
- **When a process must be selected from the ready queue, the process with the smallest next CPU burst is selected.**
- **Thus, the processes in the ready queue are sorted in next CPU burst length.**

Shortest-Job-First (SJF) Scheduling

- SJF can be nonpreemptive or preemptive.
  - 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.
- SJF is optimal – gets minimum average waiting time for a given set of processes.

SJF is provably optimal

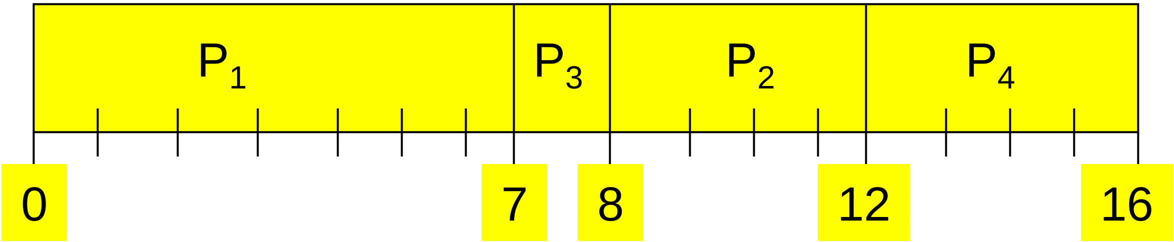


- Every time we make a short job before a long job, we reduce average waiting time.
- We may switch out of order jobs until all jobs are in order.
- If the jobs are sorted, job switching is impossible.

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)

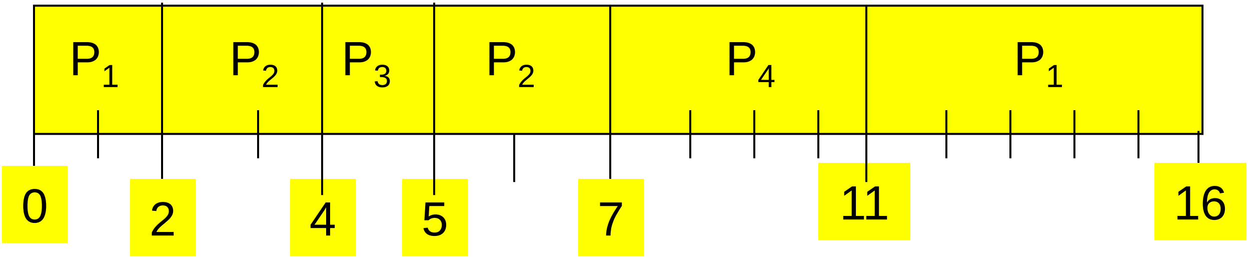


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

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 waiting time = (9 + 1 + 0 +2)/4 = 3

How do we know the Next CPU Burst?

- Without a good answer to this question, SJF cannot be used for CPU scheduling.
- We try to predict the next CPU burst!
- Can be done by using the length of previous CPU bursts —— exponential averaging (指数平均).

Exponential Averaging

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

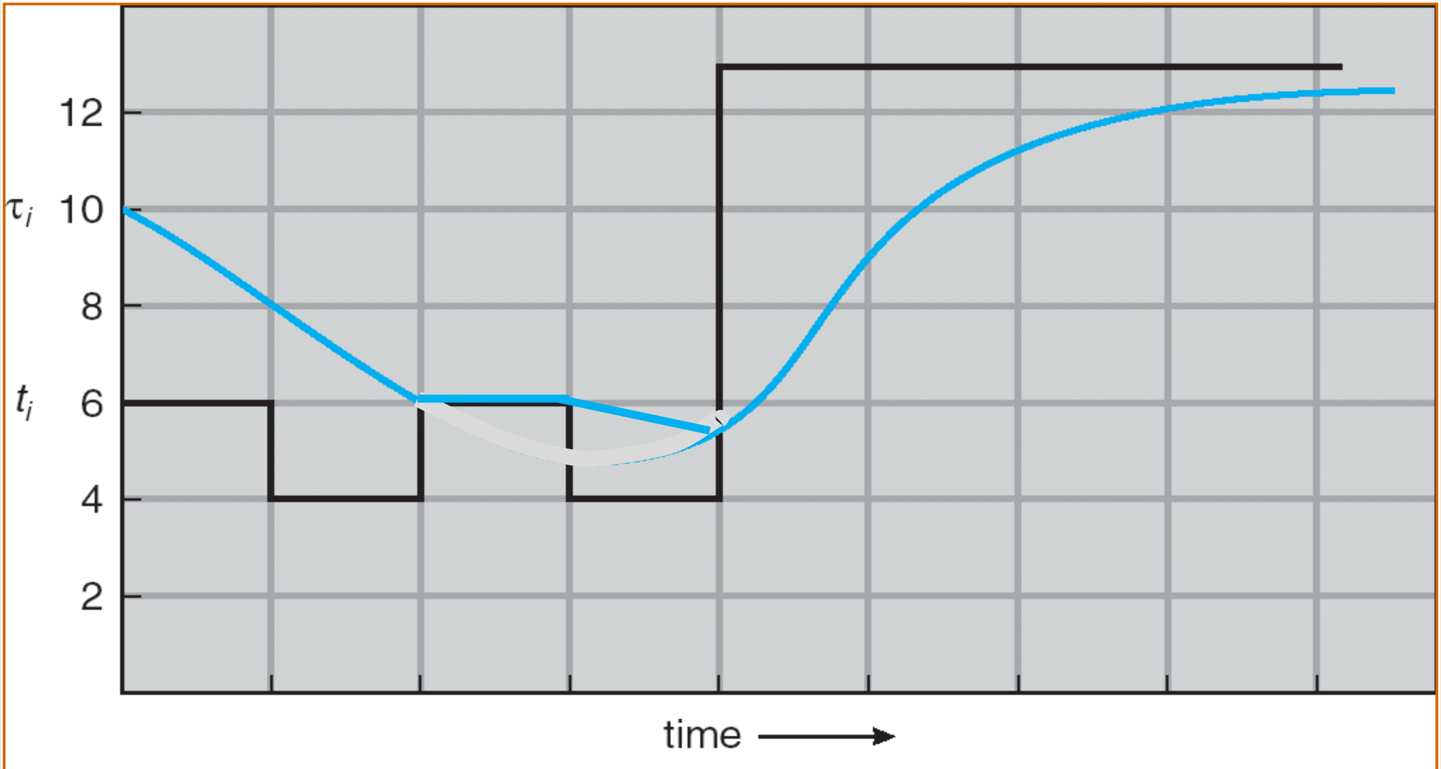
$$\tau_{n+1} = \alpha\ t_n + \left(1 - \alpha\right) \tau_n .$$

α =0， τ_{n+1} = τ_{n}, Recent history has no effect.

α =1， τ_{n+1} = t_{n}, Only the actual last CPU burst counts.

Prediction of the Length of the Next CPU Burst

$\alpha = 1/2$
$\tau_0 = 10$



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

Examples of Exponential Averaging

- **If we expand the formula, we get:**

$$
\begin{aligned}
\tau_{n+1} = \alpha\, t_n &+ (1 - \alpha)\, \alpha\, t_{n-1} + \ldots \\
&+ (1 - \alpha)^{j}\, \alpha\, t_{n-j} + \ldots \\
&+ (1 - \alpha)^{n+1}\, \tau_0
\end{aligned}
$$

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

SJF Problems

- **It is difficult to estimate the next burst time value accurately.**
- **SJF is in favor of short jobs. As a result, some long jobs may not have a chance to run at all. This is *starvation*.**

Priority Scheduling

- Each process has a ***priority***.
- Priority may be determined internally or externally:
  - **internal priority**: determined by time limits, memory requirement, number of files, and so on.
  - **external priority**: not controlled by the OS (*e.g.*, importance of the process)

Priority Scheduling

- Priorities are generally indicated by some fixed range of number, such as 0~7.
- In this text, we assume that low numbers represent high priority.
- The scheduler always picks the process (in ready queue) with the highest priority to run.
- Are FCFS and SJF the special cases of priority scheduling? (Why?)

Priority Scheduling

- Priority scheduling can be nonpreemptive or preemptive.
- With preemptive priority scheduling, if the newly arrived process has a higher priority than the running one, the latter is preempted.
- Indefinite block (or starvation) may occur: a low priority process may never have a chance to run

Aging

- **Aging is a technique to overcome the starvation problem.**
- ***Aging*: gradually increases the priority of processes that wait in the system for a long time.**
- **Example:**
  - **If 0 is the highest (*resp*., lowest) priority, then we could decrease (*resp*., increase) the value of the priority of a waiting process by 1 every fixed period (*e.g*., every minute).**

Round Robin (RR)

- RR is designed especially for time-sharing systems.
- RR is similar to FCFS, except that each process is assigned a time quantum/slice.
- All processes in the ready queue is a FIFO list. When the CPU is free, the scheduler picks the first and lets it run for one time quantum.

Round Robin (RR)

- **If the process uses CPU for less than one time quantum, it will release the CPU voluntarily.**
- **Otherwise, when one time quantum is up, that process is preempted by the scheduler and moved to the tail of the list.**

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

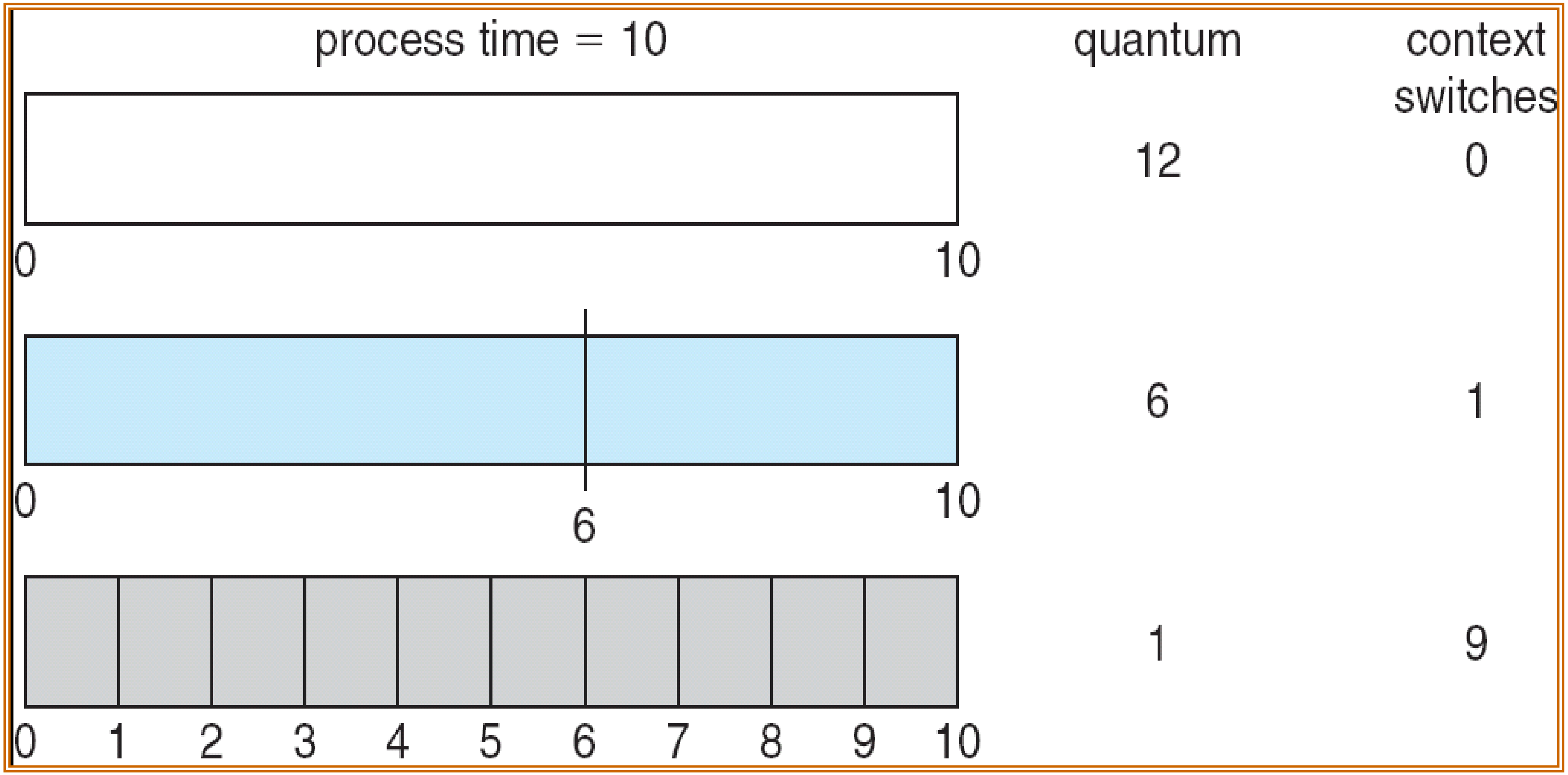
0 20 37 57 77 97 117 121 134 154 162

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

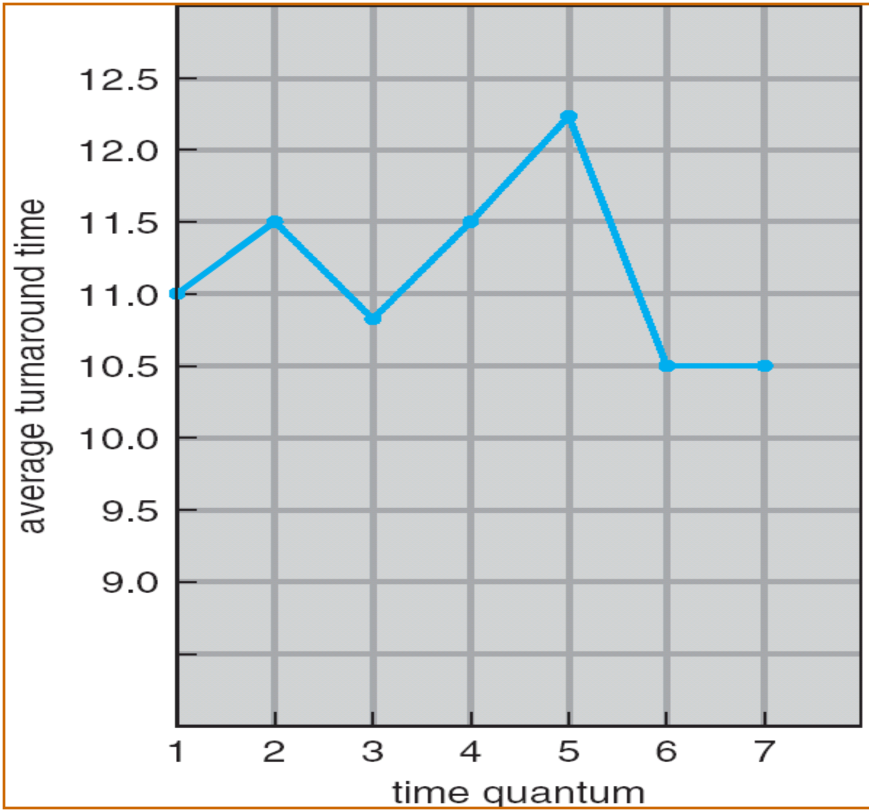
RR Scheduling Issues

- If time quantum is too large, RR reduces to FCFS
- If time quantum is too small, RR becomes processor sharing
- Context switching may affect RR’s performance
  - Shorter time quantum means more context switches
- Turnaround time also depends on the size of time quantum.
- In general, 80% of the CPU bursts should be shorter than the time quantum

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum

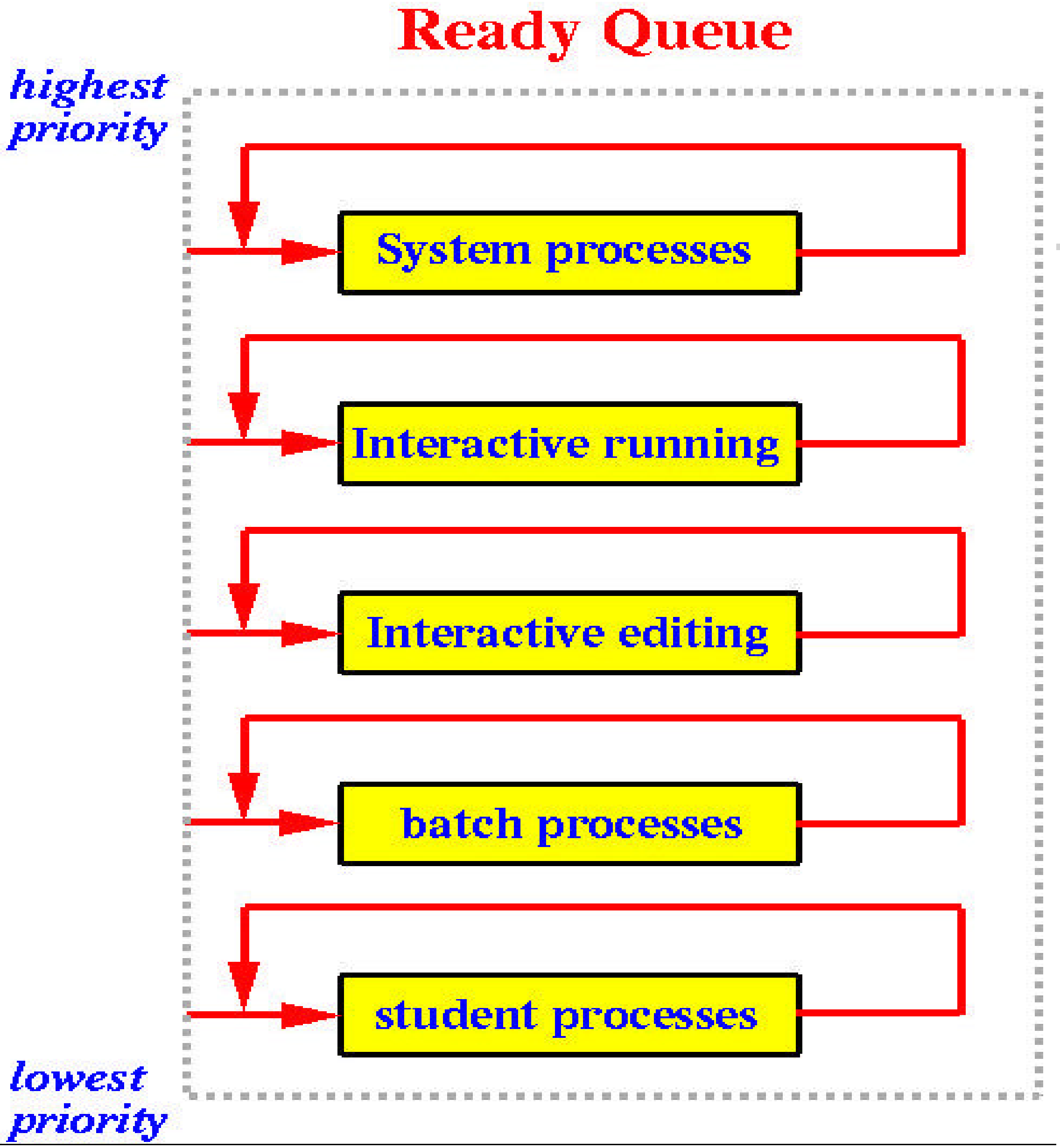


process	time
P_1	6
P_2	3
P_3	1
P_4	7

With the increasement of the time quantum， the average turnaround time is not necessarily reduced.

Multilevel Queue

- **Ready queue is partitioned into separate queues**
  - **foreground (interactive)**
  - **background (batch)**
- **Each process is assigned permanently to one queue based on some properties of the process (*e.g.*, memory usage, priority, process type)**
- **Each queue has its own scheduling algorithm**
  - **foreground – RR**
  - **background – FCFS**



- A process P can run only if all queues above the queue that contains P are empty.
- When a process is running and a process in a higher priority queue comes in, the running process is preempted.

Multilevel Queue

- Scheduling must be done between the queues.
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR, 20% to background in FCFS

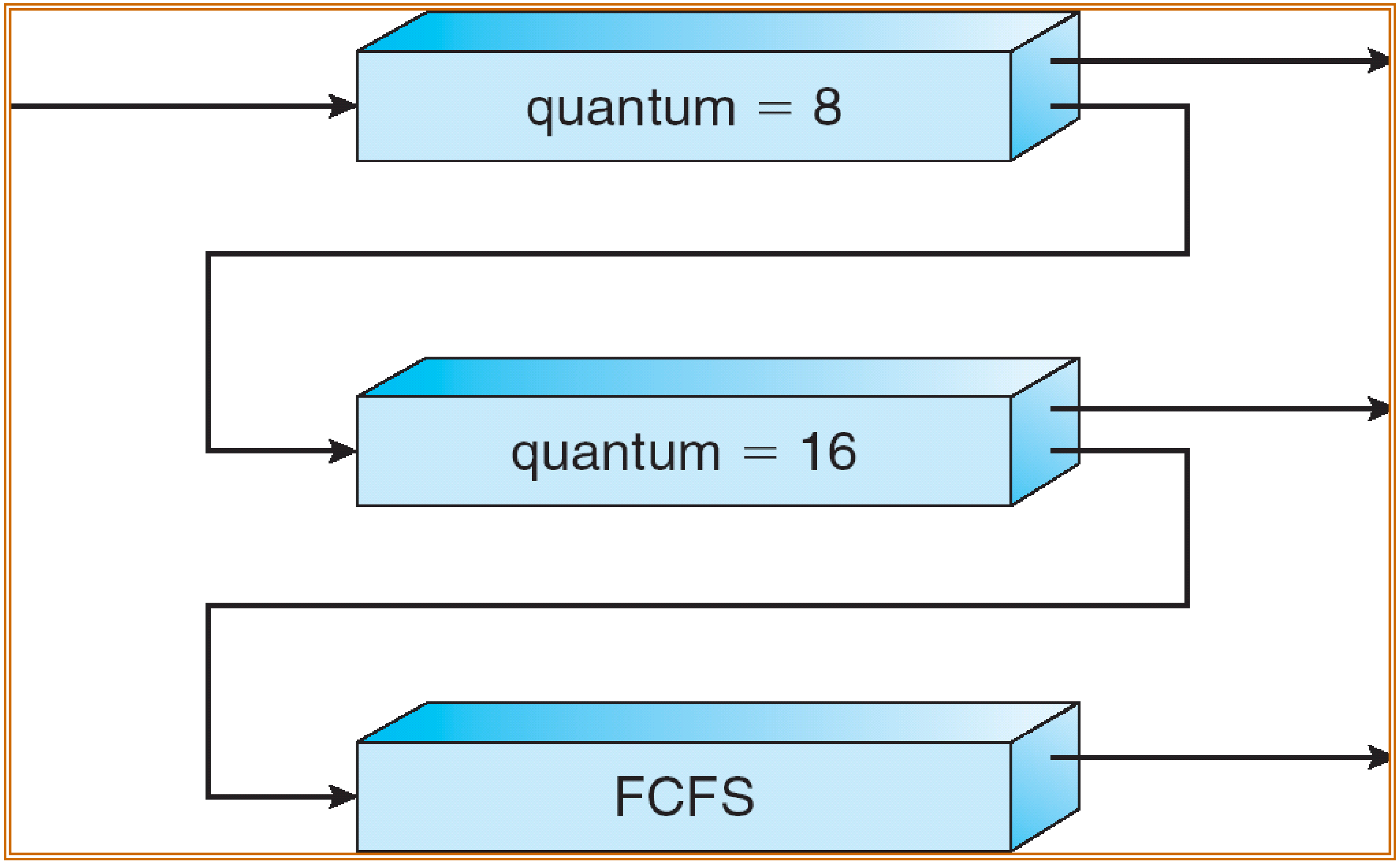
Multilevel Feedback Queue

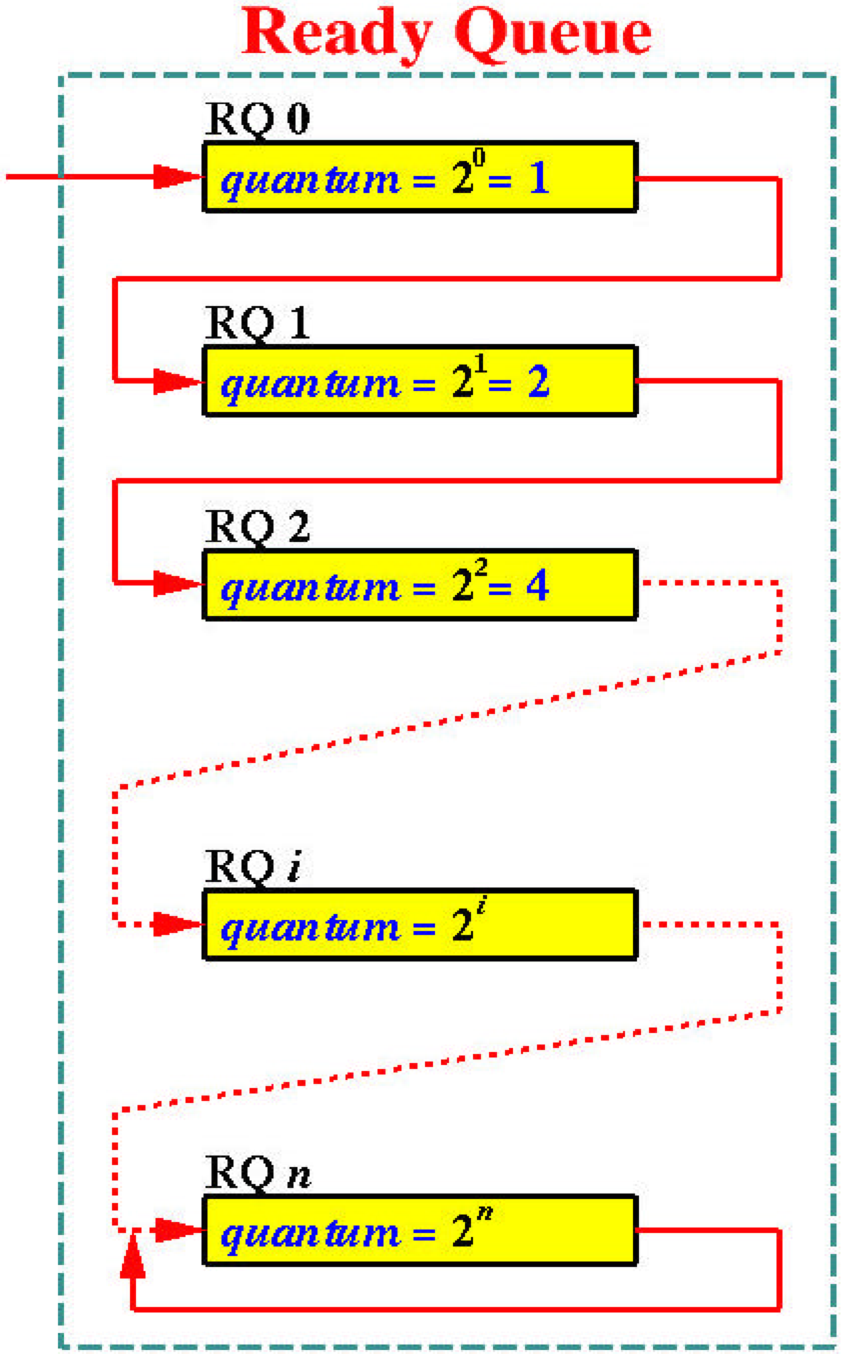
- ***Multilevel queue with feedback scheduling*** **is similar to multilevel queue; however, it allows processes to move between queues.**
  - **aging can be implemented this way**
- **If a process uses more (*resp*., less) CPU time, it is moved to a queue of lower (*resp*., higher) priority.**
- **As a result, I/O-bound (*resp*., CPU-bound) processes will be in higher (*resp*., lower) priority queues.**

Example of Multilevel Feedback Queue

- **Three queues:**
  - **Q_0 – time quantum 8 milliseconds**
  - **Q_1 – time quantum 16 milliseconds**
  - **Q_2 – FCFS**
- **Scheduling**
  - **A new job enters Q_0 which is served RR. When it gains CPU, job runs for 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to Q_1.**
  - **At Q_1 job is again served RR and runs for 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.**

Multilevel Feedback Queues





- Processes in queue *i* have time quantum 2^i
- When a process' behavior changes, it may be placed (*i.e.*, promoted or demoted) into a difference queue.
- Thus, when an CPU-bound process starts to use more CPU, it may be demoted to a lower queue

Multilevel Feedback Queue

- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Thread Scheduling
- Algorithm Evaluation
- Operating System Examples

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- *Homogeneous processors* within a multiprocessor.
- *Load Balancing* — keep the workload evenly distributed across all processors.
  - Push migration
  - Pull migration

Multiple-Processor Scheduling

- *Asymmetric multiprocessing* – only one processor accesses the system data structures, alleviating the need for data sharing.
- *Symmetric multiprocessing (SMP)* – each processor is self-scheduling.
- For SMP, two processors do not choose the same process.

Multiple-Processor Scheduling

- ***Processor Affinity***
  - **Most SMP systems try to avoid migration of processes from one processor to another.**
  - **Soft Affinity & Hard Affinity**

Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Thread Scheduling
- Algorithm Evaluation
- Operating System Examples

Real-Time Scheduling

- *Hard real-time systems* – required to complete a critical task within a guaranteed amount of time.
  - The scheduler either admits a process and guarantees that the process will complete on-time, or reject the request (*resource reservation*).
  - This is almost impossible if the system has secondary storage and virtual memory because these subsystems can cause unavoidable delay.
  - Hard real-time systems usually have special software running on special hardware.

Real-Time Scheduling

- ***Soft real-time systems*** **– requires that critical processes receive priority over noncritical ones.**
  - **It is easily doable within a general system.**
  - **It could cause an unfair resource allocation and longer delay (starvation) for noncritical tasks.**
  - **The CPU scheduler must prevent aging to occur. Otherwise, critical tasks may have lower priority.**
  - **The dispatch latency must be small.**

Priority Inversion

- What if **a high-priority process needs to access the data that is currently being accessed by a low-priority process?**
  - The high-priority process is blocked by the low-priority process.
  - This is ***priority inversion***.

Priority Inversion

- This can be solved with ***priority-inheritance protocol***.
  - All processes, including the one that is accessing the data, inherit the high priority until they are done with the resource.
  - When they finish, their priority values revert back to the original values.

Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Thread Scheduling
- Algorithm Evaluation
- Operating System Examples

Thread Scheduling

- User-level threads
  - managed by a thread library
- Kernel-level threads
  - scheduled by OS
- To run on a CPU, user-level threads must ultimately be mapped to an associated kernel-level thread.

Thread Scheduling

- Local Scheduling – User-level Thread
  - The threads library decides which thread to put onto an available LWP
  - Process-contention Scope (PCS)
- Global Scheduling – Kernel-level Thread
  - The kernel decides which kernel thread to run next
  - System-contention Scope (SCS)

Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Thread Scheduling
- Algorithm Evaluation
- Operating System Examples

Algorithm Evaluation

- Deterministic modeling (Analytic evaluation)
  - taking a particular predetermined workload and computing the performance of each algorithm for that workload.
  - describing scheduling algorithms and providing examples.
- Queueing models
  - Providing the approximations of real system
- Simulations
  - Providing the approximations of real system
- Implementation

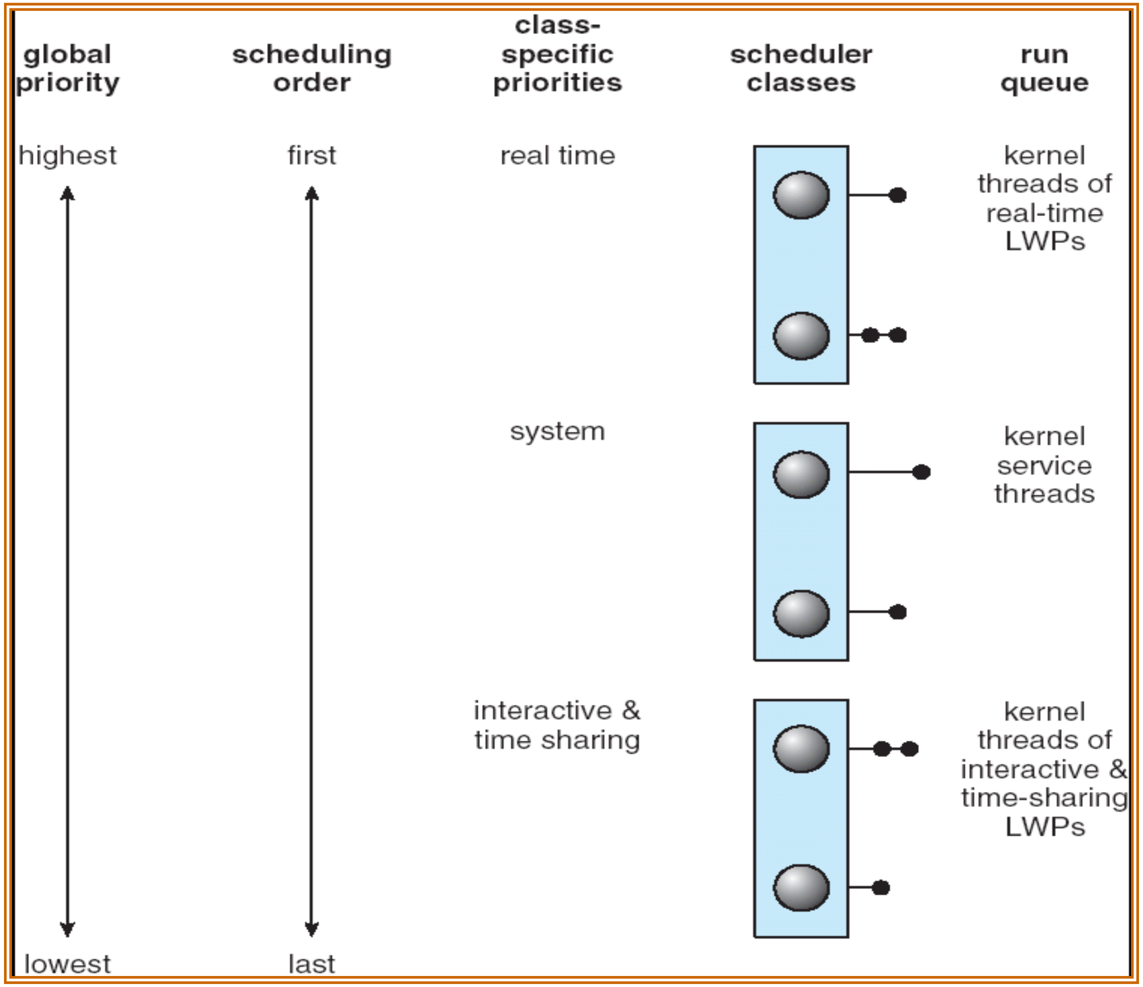
Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Thread Scheduling
- Algorithm Evaluation
- Operating System Examples

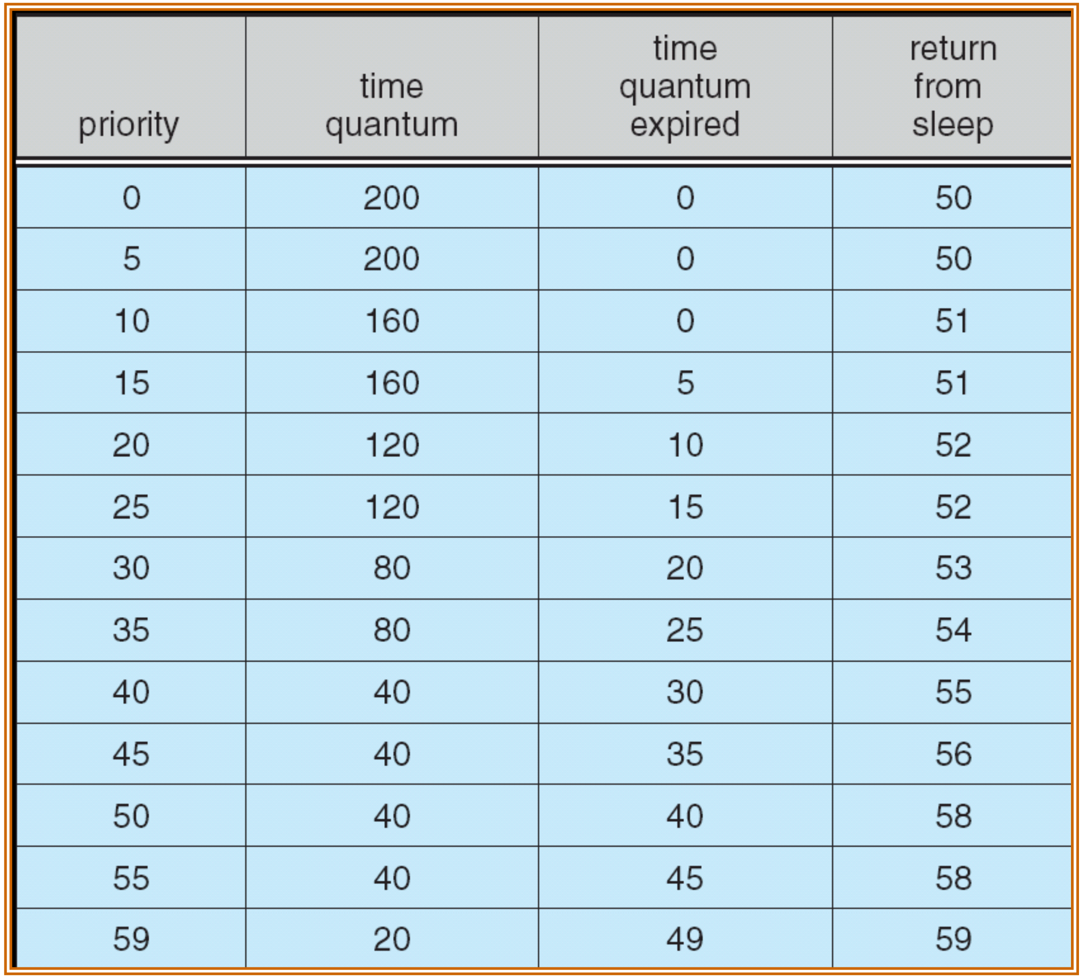
Solaris 2 Scheduling

- Scheduling threads using preemptive and priority-based scheduling algorithms. (Real time, System, Time sharing, Interactive)
- The default scheduling class for a process is time sharing —— Multilevel Feedback Queue

Solaris 2 Scheduling


global priority
scheduling order
class-specific priorities
scheduler classes
run queue
highest
first
real time
kernel threads of real-time LWPs
system
kernel service threads
interactive & time sharing
kernel threads of interactive & time-sharing LWPs
lowest
last

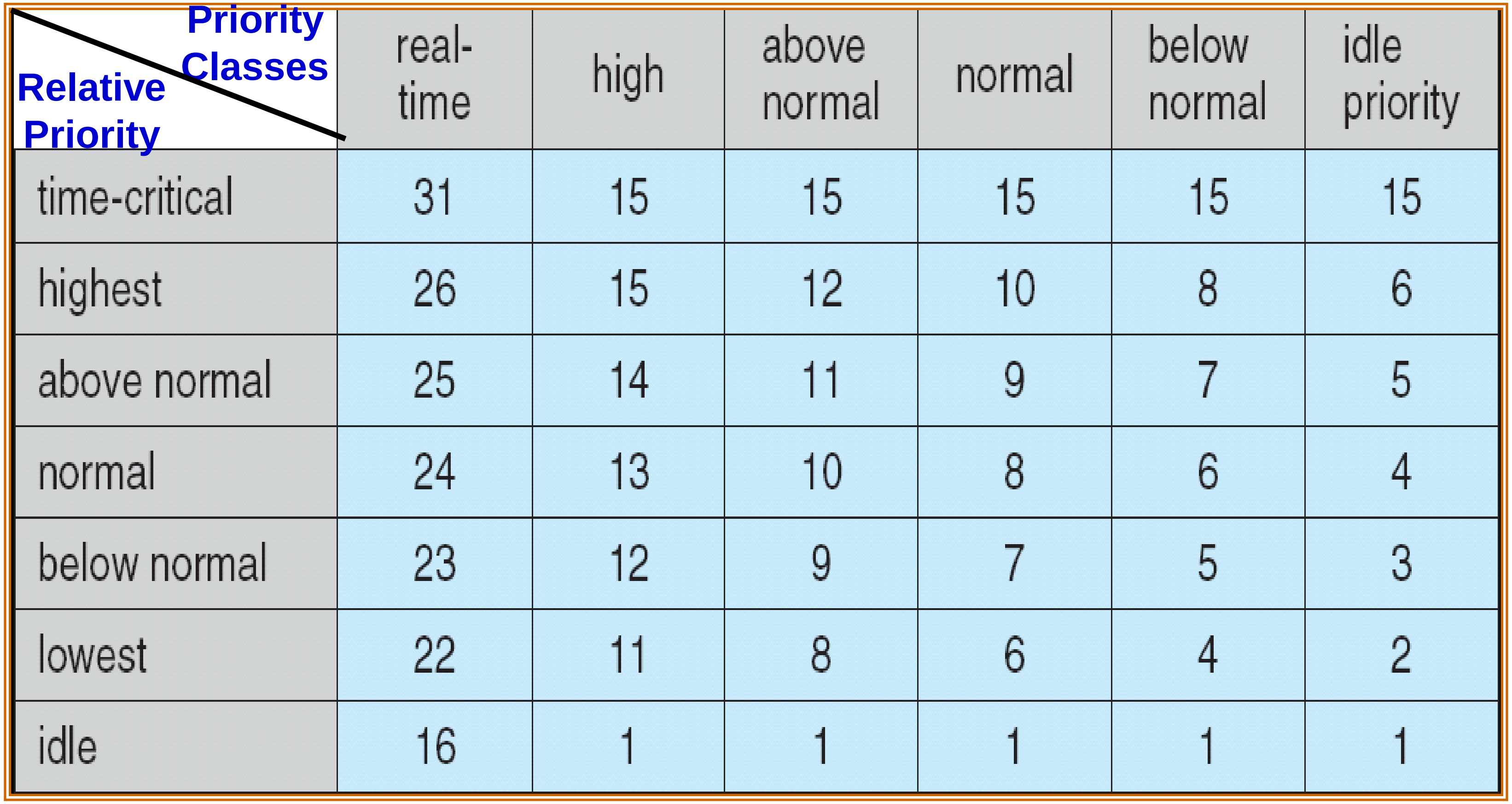
Solaris Dispatch Table



priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

Windows XP Priorities

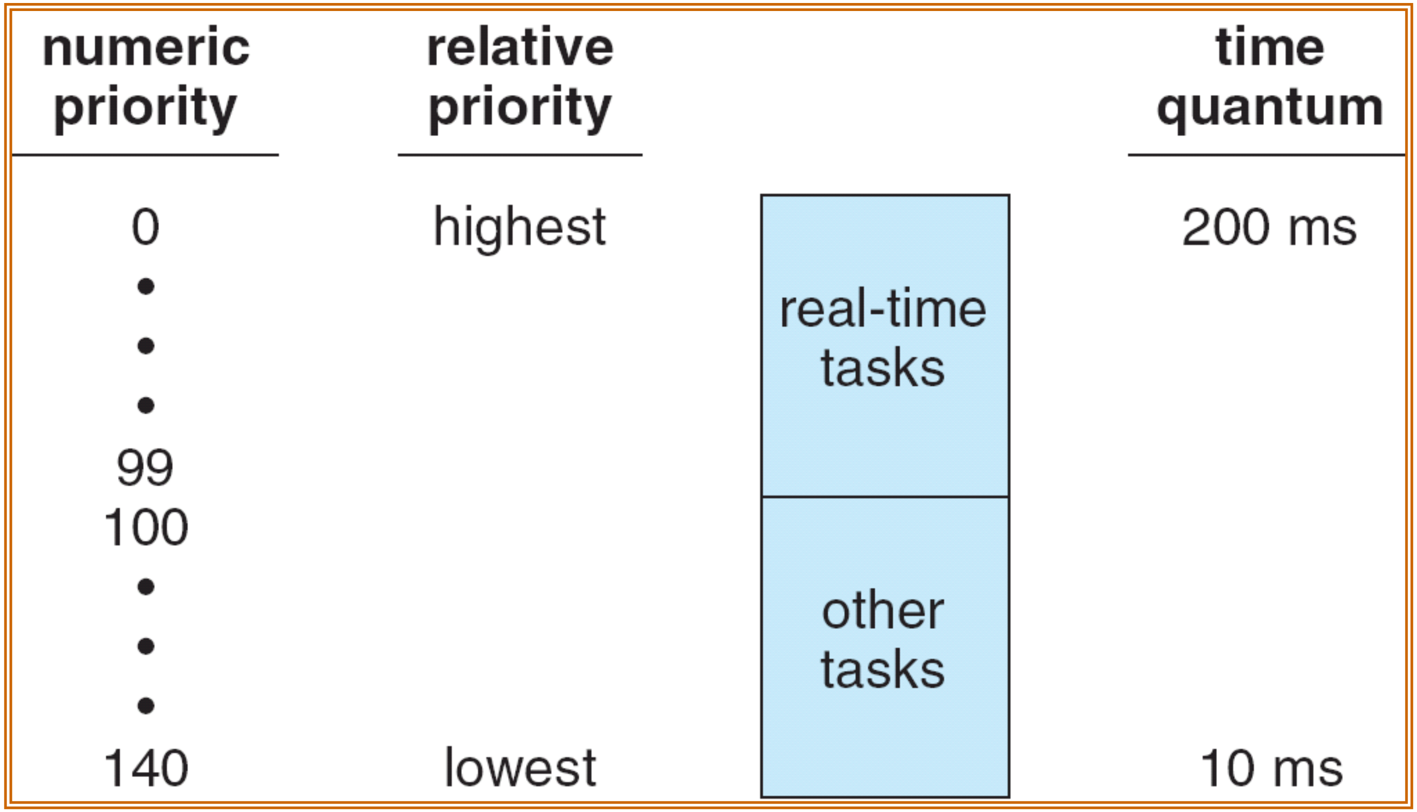
- **preemptive** and **priority-based** scheduling algorithms



Priority Classes Relative Priority	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Linux Scheduling

- **preemptive and priority-based scheduling algorithms**



numeric priority	relative priority		time quantum
0	highest	real-time tasks	200 ms
•			
•			
•			
99			
100		other tasks	
•			
•			
•			
140	lowest		10 ms

Linux Scheduling

- 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 runable processes have credit = 0, recrediting occurs based on factors including priority and history
- Real-time
  - Soft real-time —— FCFS and RR
  - Highest priority process always runs first

作业2（P187 课后习题5.4）

- Consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5 all at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum=1).

b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

c. What is the waiting time of each process for each of the scheduling algorithms in part a?

d. Which of the algorithms in part a results in the minimum average waiting time (over all processes)?