0117401: Operating System
计算机原理与设计

Chapter 5: CPU scheduling

陈香兰
xlanchen@ustc.edu.cn
http://staff.ustc.edu.cn/~xlanchen

Computer Application Laboratory, CS, USTC @ Hefei
Embedded System Laboratory, CS, USTC @ Suzhou

May 31, 2017

温馨提示：

为了您和他人的工作学习，
请在课堂上关机或静音。

不要在课堂上接打电话。

Chapter Objectives

- To **introduce** CPU scheduling.
- To describe various CPU-scheduling **algorithms**.
- To discuss **evaluation crieria** for selecting a CPU-scheduling algorithm for a particular system.

提纲——CPU scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- OS examples
- Algorithm Evaluation
- 小结和作业

Outline

Basic Concepts

- Scheduling is a fundamental OS function.
  - Almost all computer resources are scheduled before use.
  - CPU scheduling is the basis of multiprogrammed OSes.

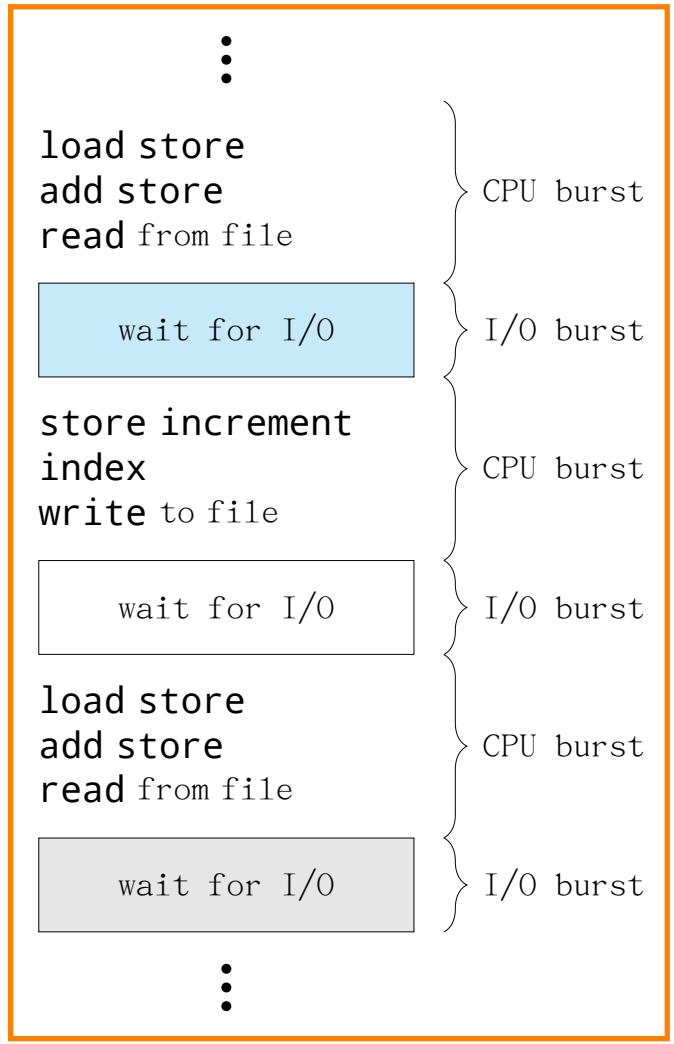
Objective of multiprogramming

- Maximum CPU utilization

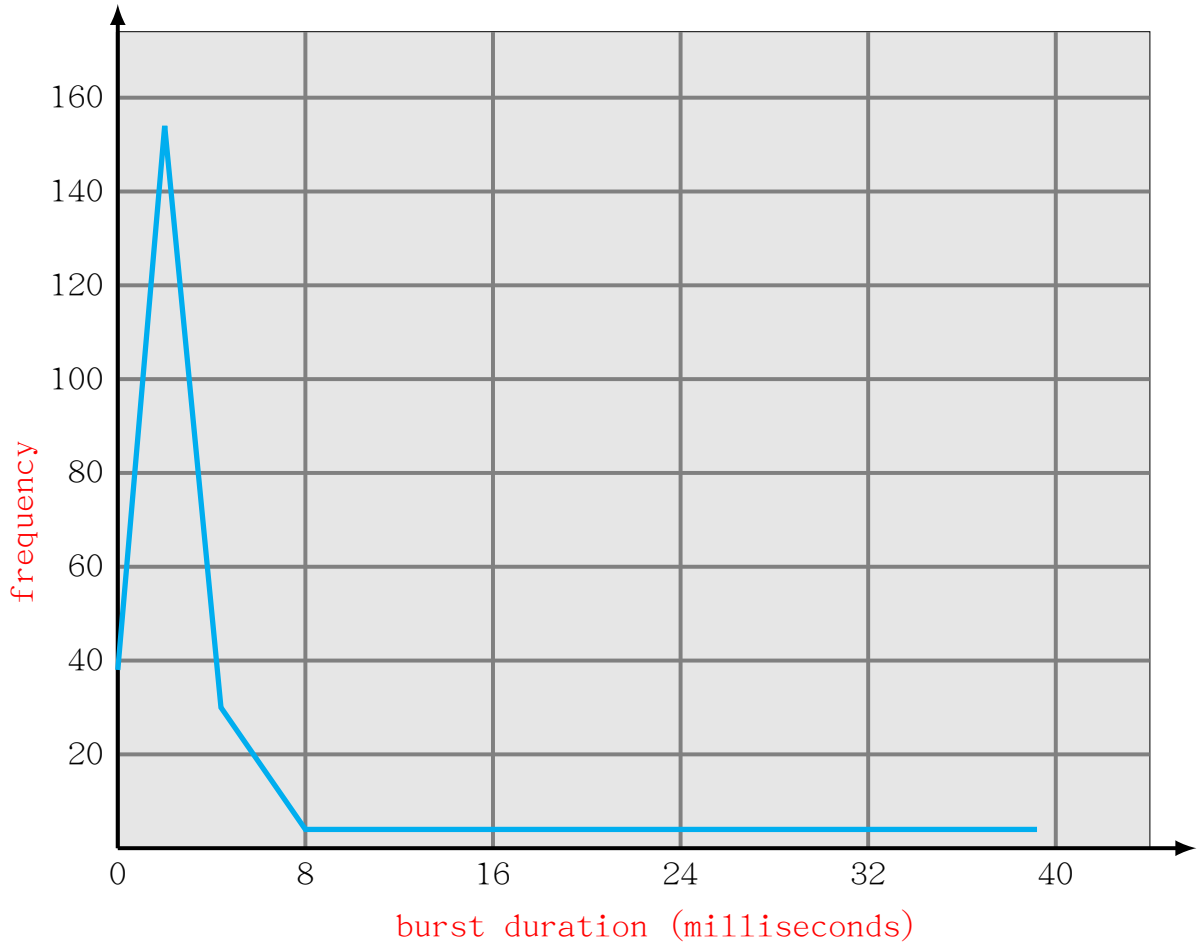
Outline

Basic Concepts: CPU-I/O Burst Cycle

- A **property** of process :
  CPU-I/O Burst Cycle
- Process execution consists of a cycle of CPU execution and I/O wait
- Alternating Sequence of CPU And I/O Bursts
- Begin and end with a CPU burst
- Process execution
  = n (CPU execution + I/O wait)
  + CPU execution



CPU burst distribution



Histogram of CPU-burst Times

Outline

CPU Scheduler

- **CPU scheduler** (Short-term Scheduler) **selects** a process from the processes in memory that are ready to execute and **allocates** the CPU to the process
- **Ready Queue** could be:
  - a FIFO Queue?
  - a priority queue?
  - a tree?
  - an unordered linked list?

Outline

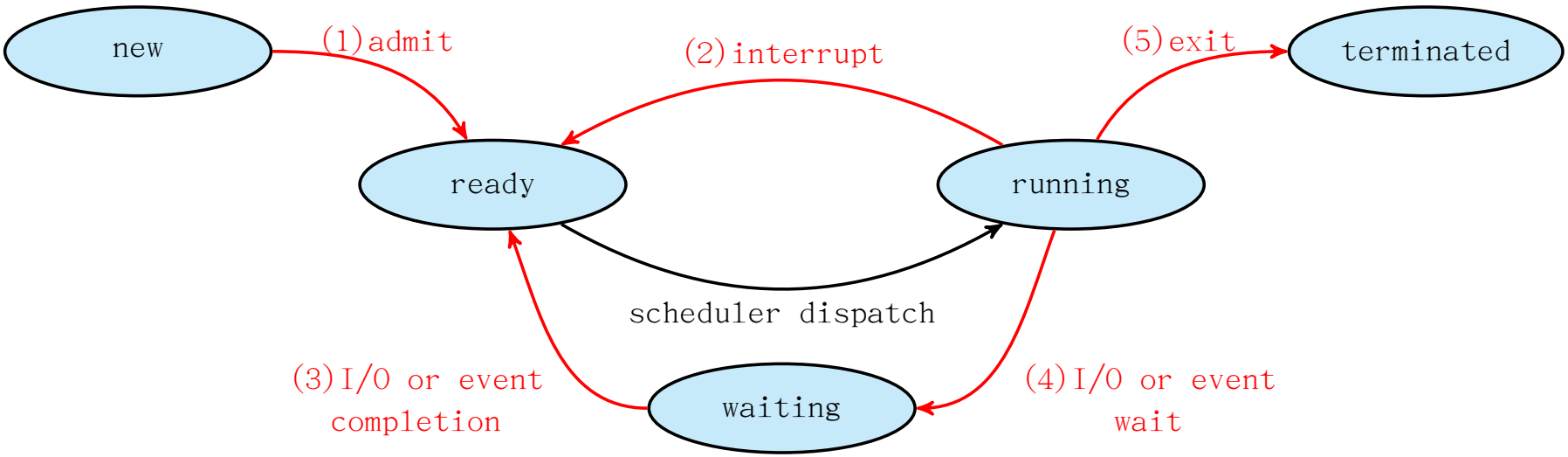
Preemptive Scheduling I

- CPU scheduling decisions may take place when a process:
  1. Switches from **new to ready** state
  2. Switches from **running to ready** state
  3. Switches from **waiting to ready** state
  4. Switches from **running to waiting** state
  5. **Terminates**

  For 4 & 5, must schedule;

  For 1 & 2 & 3, schedule? VS. not schedule?

Preemptive Scheduling II



Two scheduling scheme:

1. **nonpreemptive(非抢占式)**: only 4 & 5
   - Windows 3.x
   - before Mac OS X
2. otherwise **preemptive(抢占式)**
   - Windows 95 & ...
   - Mac OS X

Preemptive Scheduling III

- usually needs a hardware **timer**, **synchronization** overhead

Two processes sharing data

- If one process is preempted while it is updating the data,
  data is in an **inconsistent(不一致)** state

COST for preemption

1. needs special HW, for example, a **timer**.
2. **synchronization** overhead with shared data.

Preemption of the OS kernel

- What happens if the process is preempted in the middle of some activities that changes important kernel data?

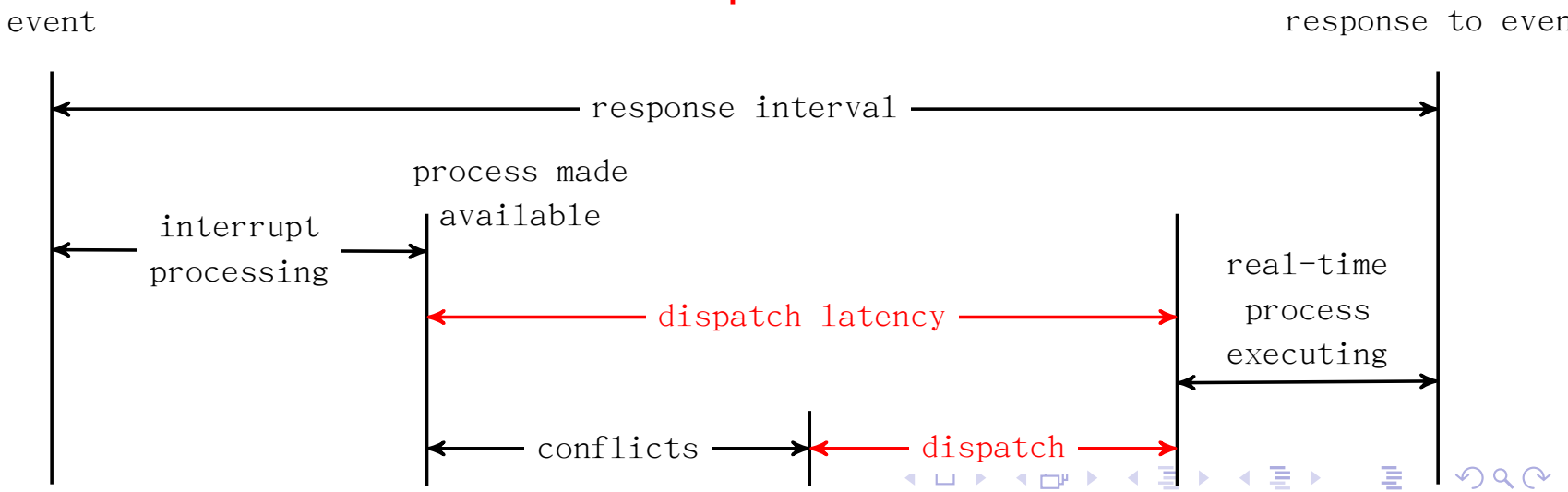
- preemptive kernel VS. nonpreemptive kernel?
- Interrupt affected code VS normal kernel code?

- new mechnisms are needed, such as
  - disable interrupt
  - some synchronization mechnisms

Outline

Dispatcher

- Dispatcher module **gives control of the CPU** to the process selected by the cpu scheduler; this involves:
  1. switching context
  2. switching to user mode
  3. jumping to the proper location in the user program to continue the execution of that program
- **Dispatch latency** — time it takes for the dispatcher to stop one process and start another running
  - **SHOULD be as fast as possible**



Outline

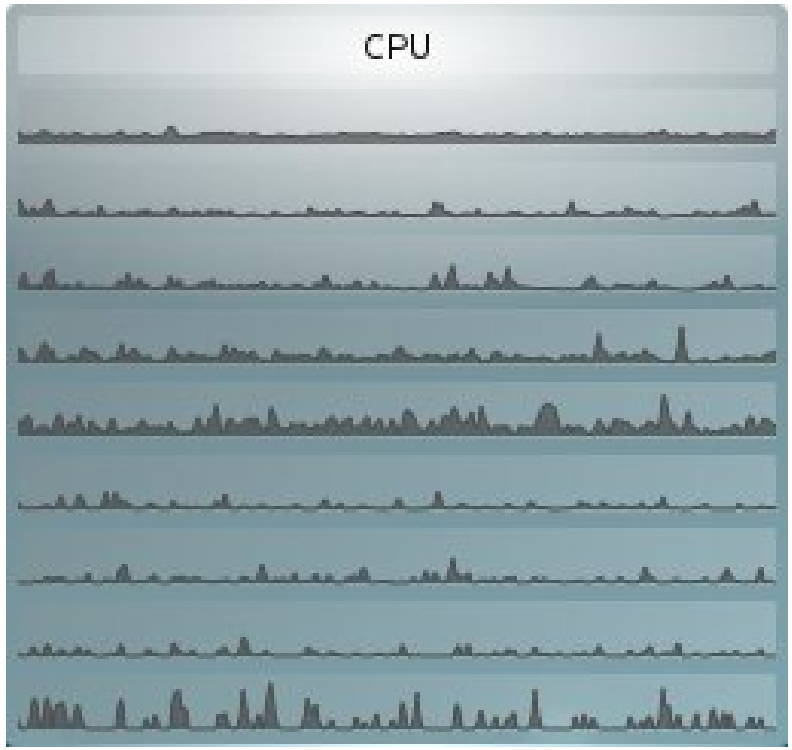
Outline

Scheduling Criteria

1. **CPU utilization** (CPU 利用率)
2. **Throughput** (吞吐率)
3. **Turnaround time** (周转时间)
4. **Waiting time** (等待时间)
5. **Response time** (响应时间)

Scheduling Criteria

1. **CPU utilization** (CPU 利用率)— keep the CPU as busy as possible
   - conceptually: $0\% \sim 100\%$; in a real system: $40\% \sim 90\%$



4核8线程编译Linux内核时的CPU利用率情况（0-7，总）

2. **Throughput** (吞吐率)
3. **Turnaround time** (周转时间)
4. **Waiting time** (等待时间)

Scheduling Criteria

1. **CPU utilization** (CPU 利用率)
2. **Throughput** (吞吐率)— # of processes that complete their execution per time unit
   - different from one process set to another process set
   - **for long processes:** may be 1 process per hour
   - **for short transactions:** may be 10 processes per second
3. **Turnaround time** (周转时间)
4. **Waiting time** (等待时间)
5. **Response time** (响应时间)

Scheduling Criteria

1. **CPU utilization** (CPU 利用率)
2. **Throughput** (吞吐率)
3. **Turnaround time** (周转时间) — amount of time to execute a particular process
   - **from** the time of **submission** of a process **to** the time of **completion**
     = the periods spent **waiting to get into memory**, **waiting in the ready queue**, **executing** on the CPU, and **doing I/O**.
4. **Waiting time** (等待时间)
5. **Response time** (响应时间)

Scheduling Criteria

1. **CPU utilization** (CPU 利用率)
2. **Throughput** (吞吐率)
3. **Turnaround time** (周转时间)
4. **Waiting time** (等待时间)— amount of time a process has been **waiting in the ready queue**
5. **Response time** (响应时间)

Scheduling Criteria

1. **CPU utilization** (CPU 利用率)
2. **Throughput** (吞吐率)
3. **Turnaround time** (周转时间)
4. **Waiting time** (等待时间)
5. **Response time** (响应时间)— amount of time it takes **from** when a request was **submitted until the first response** is produced, not output
   - for time-sharing environment

Optimization Criteria

- Maximize?
  - CPU utilization
  - throughput
- Minimize?
  - turnaround time
  - waiting time
  - response time
- Average?
- Stability?

different from system to system.

Outline

Outline

FCFS Scheduling

- First-Come, First-Served(先来先服务)
  - nonpreemptive(非抢占)
- Implementation:
  - Normal Queue: FIFO Queue
  - ordered by request time
  - linked list
  - Insert: linked to the tail of the queue
  - scheduling: removed from the head of the queue

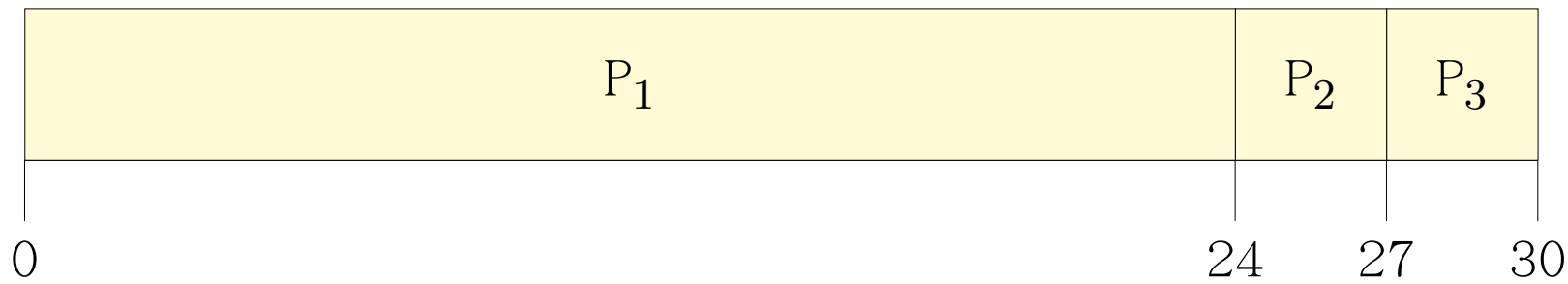
Example of FCFS Scheduling

- Suppose that the processes arrive in the **order**:

  P1 , P2 , P3

Process	BurstTime(ms)
P1	24
P2	3
P3	3

  - The **Gantt Chart(甘特图)** for the schedule is:



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

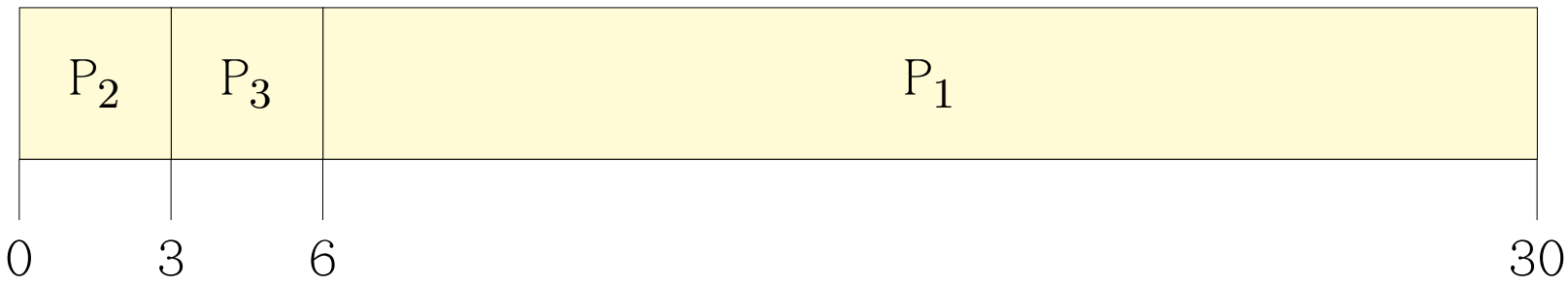
Example of FCFS Scheduling II

- Suppose that the processes arrive in the order

  P2 , P3 , P1

Process	BurstTime(ms)
P1	24
P2	3
P3	3

  - The Gantt chart(甘特图) for the schedule is:



  - Waiting time for P1 = 6; P2 = 0; P3 = 3
  - Average waiting time: $(6+0+3)/3 = 3$

MUCH BETTER THAN PREVIOUS CASE!

Convoy effect（护航效应；护卫效应）

Convoy effect（护航效应；护卫效应）

- all the other processes wait for the one big process to get off the CPU
- ≡short process behind long process

example situation:

- one CPU-bound process
- many I/O-bound processes

Outline

SJF Scheduling

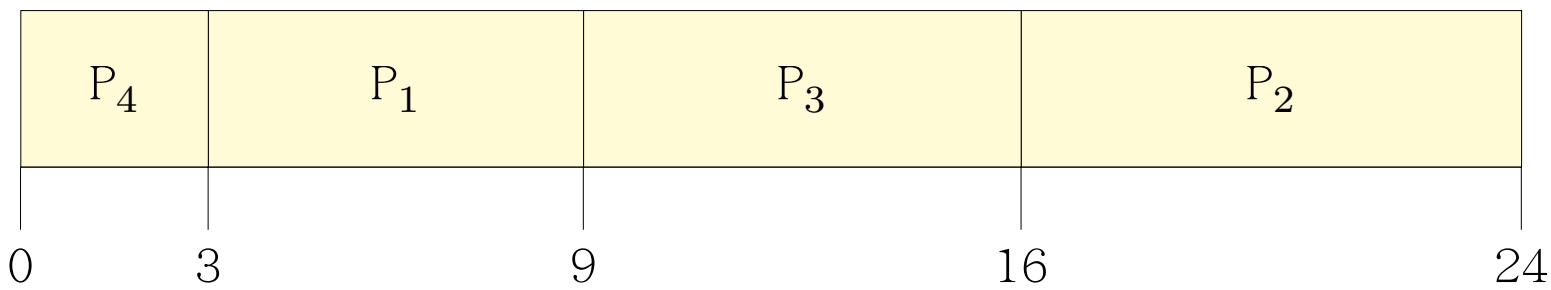
- Shortest-Job-First(短作业优先)
  Shortest-Next-CPU-Burst algorithm
  - Associate with each process the length of its next CPU burst.
  - Schedule the process with the shortest time.

SJF Scheduling

- SJF scheduling example

Process	BurstTime(ms)
P1	6
P2	8
P3	7
P4	3

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



1. **Waiting time** for P1 = 3; P2 = 16; P3 = 9; P4 = 0
2. **Average waiting time**: $(3 + 16 + 9 + 0)/4 = 7$

If FCFS, **average waiting time**:
$(0 + 6 + 14 + 21)/4 = 10.25$

SJF Scheduling

SJF is optimal(最优的)
— gives minimum average waiting time(最小平均等待时间) for a given set of processes

SJF scheduling schemes

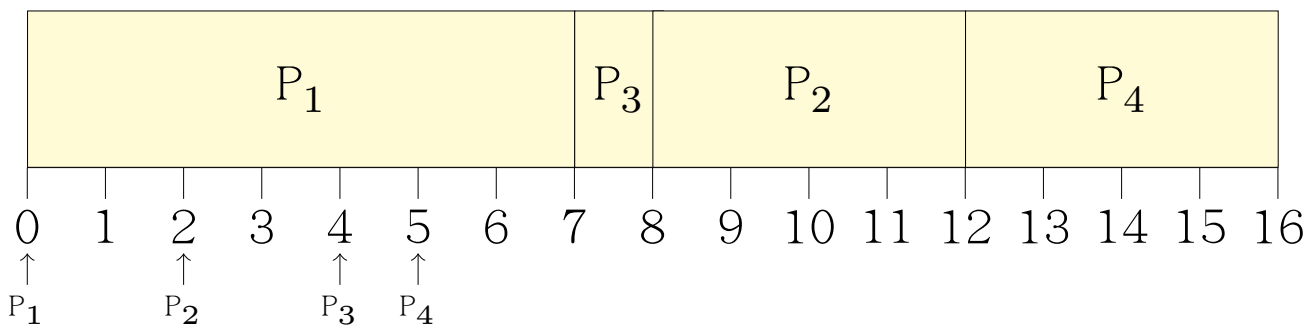
- Two schemes:
  1. **nonpreemptive**
     — once CPU given to the process it cannot be preempted until completes its CPU burst
  2. **preemptive**
     — if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the **Shortest-Remaining-Time-First** (SRTF)

SJF scheduling schemes

1. Example of Non-Preemptive SJF

Process	ArrivalTime	BurstTime(ms)
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

- The **Gantt chart** for SJF (non-preemptive)



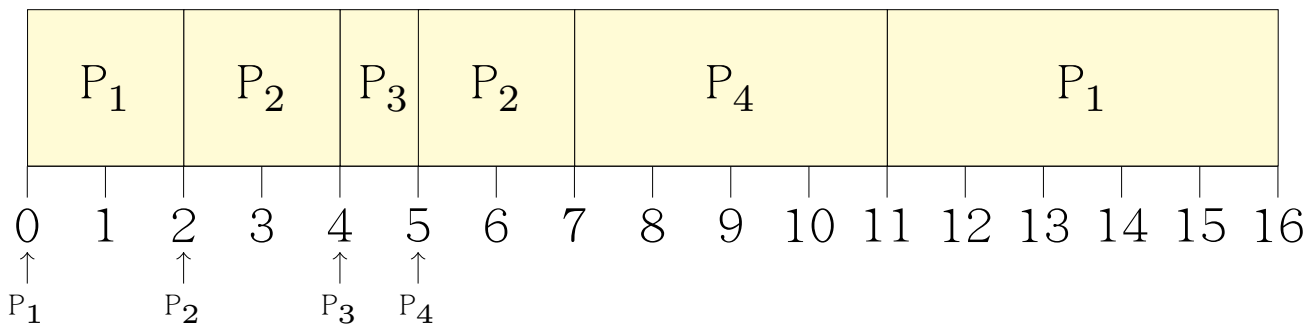
- **Average waiting time**: $(0+6+3+7)/4 = 4$

SJF scheduling schemes

2. Example of Preemptive SJF

Process	ArrivalTime	BurstTime(ms)
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

- The **Gantt chart** for SJF (**preemptive**)



- **Average waiting time**: $((11-2)+(5-4)+0+(7-5))/4 = 3$

Determining Length of Next CPU Burst

- For job scheduling:

  depend on user?

- For CPU scheduling:

  can only **estimate** the length

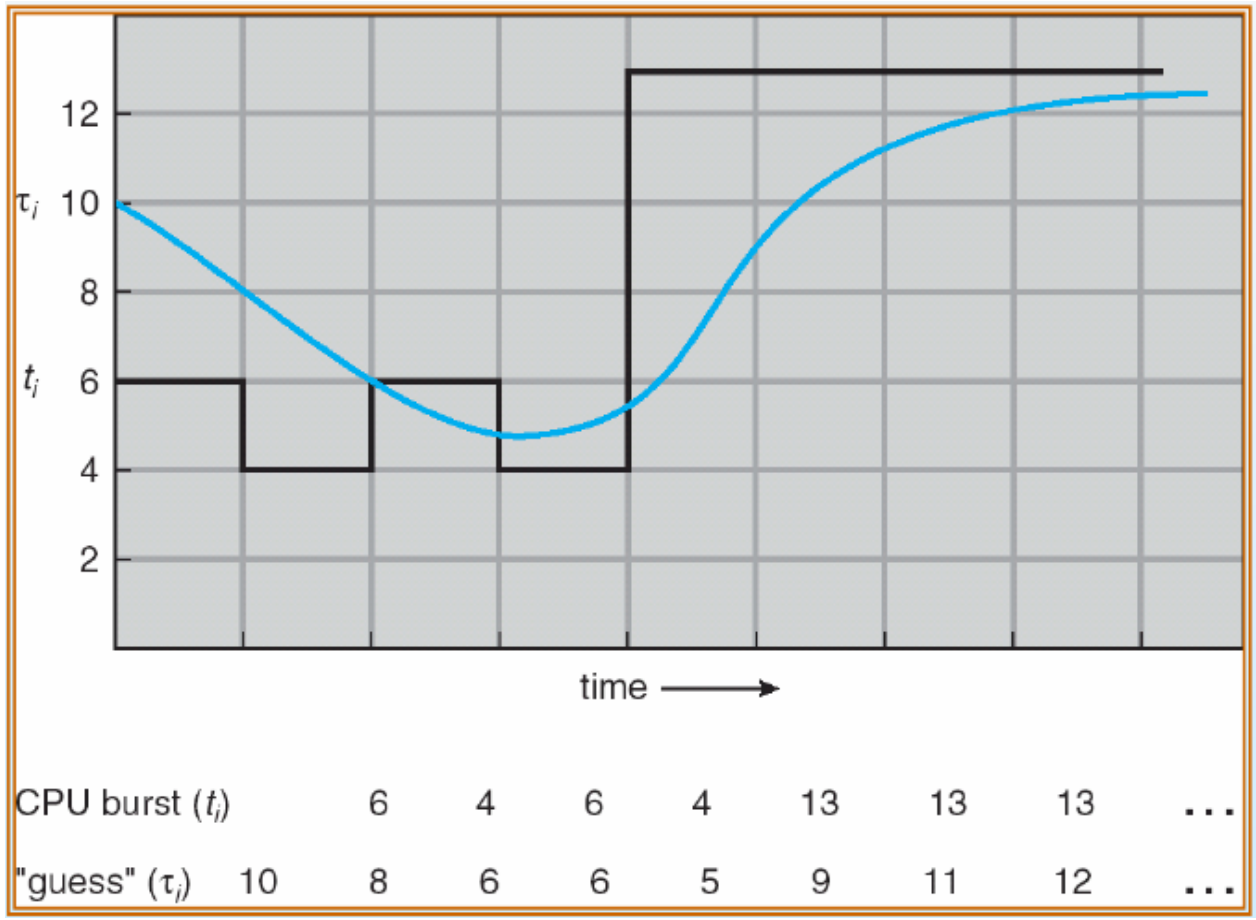
- Example: by using the length of previous CPU bursts, using **exponential averaging(指数平均)**
  1. t_n =actual length of n^{th} CPU burst
  2. τ_{n+1}= predicted value for the next CPU burst
  3. $\alpha, 0 \leq \alpha \leq 1$
  4. Define: $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\tau_n + (1-\alpha)\,\alpha\tau_{n-1} + \cdots + (1-\alpha)^j\,\alpha\tau_{n-j} + \cdots + (1-\alpha)^{n+1}\,\tau_0$$

Since $0 \leq \alpha,\ 1-\alpha \leq 1$, each successive term has less weight than its predecessor

Determining Length of Next CPU Burst

- Prediction of the Length of the Next CPU Burst
  - Example: $\alpha = 1/2;\ \tau_0 = 10$



Determining Length of Next CPU Burst

- Examples of Exponential Averaging
  - if $\alpha = 0$
    - $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n = \mathbf{0} \cdot t_n + \tau_n = \tau_n$
    - Recent history does not count
  - if $\alpha = 1$
    - $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n = t_n + \mathbf{0} \cdot \tau_n = t_n$
    - Only the actual last CPU burst counts

Outline

Priority(优先级) Scheduling

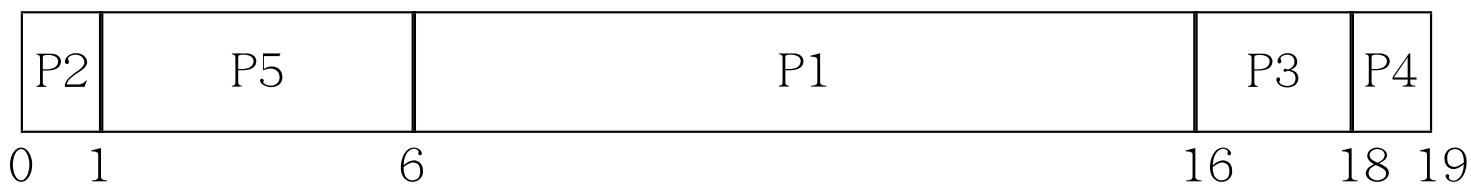
- A **priority number(优先数)** is associated with each process
  - **priority number(优先数)** VS. **priority(优先级)**
  - usually an integer, &
    usually, **smallest integer ≡ highest priority**
- The CPU is allocated to the process with the highest priority
  - **Preemptive** VS. **Nonpreemptive**
- **SJF is a special case** of general priority scheduling where priority is the predicted next CPU burst time

Priority(优先级) Scheduling

- Example

Process	BurstTime(ms)	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

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



  - **Average waiting time**: $(6+0+16+18+1)/5 = 8.2$

Priority(优先级) Scheduling

- Problem: The determination of priority
  - internally, for example:
    - time limits, memory requirement, the number of open files, ...
  - externally, for example:
    - the importance, the type and amount of funds, the department, ...

Priority(优先级) Scheduling

- Priority Scheduling problem - **Starvation** (indefinite blocking):
  low priority processes may never execute
  - Solution - **Aging**:
    as time progresses increase the priority of the process

Example:

- priorities: 127(low)-0(high)
- the priority of a waiting process is increased by 1 every 15 minutes
- How long from 127 to 0?

Outline

Round Robin（时间片轮转，RR）Scheduling

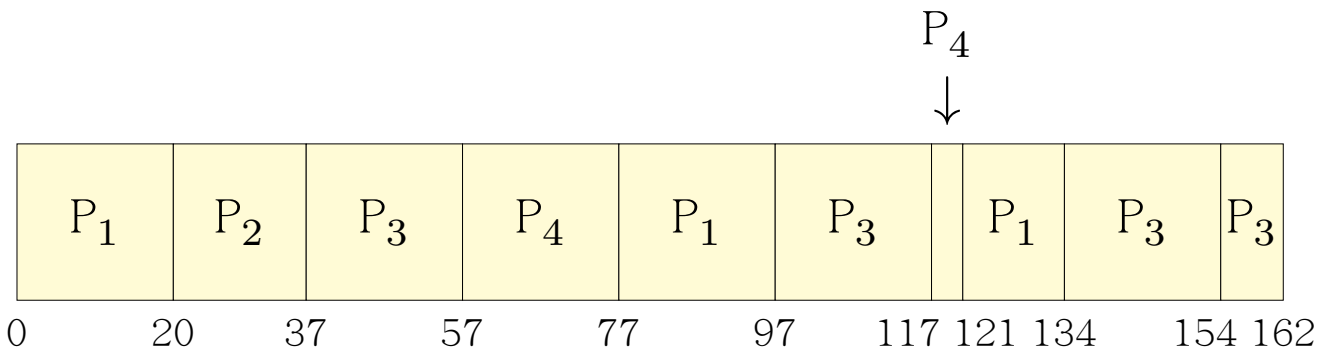
- Time quantum, time slice(时间片)
  - a small unit of CPU time
  - usually 10-100 ms
- Implementation
  - Ready queue: a FIFO circular queue
  - Each process gets 1 time quantum
  - Insert: to the tail of the queue
  - Scheduling: pick the first process; set timer; and dispatch
  - two situation:
    - CPU burst ≤ 1 time quantum
    - CPU burst > 1 time quantum.
      After this time has elapsed, the process is preempted(被抢占) and added to the end of the ready queue.

Round Robin（时间片轮转，RR）Scheduling

- Example of RR with Time Quantum = 20

Process	BurstTime
P1	53
P2	17
P3	68
P4	24

  - The **Gantt chart** is:



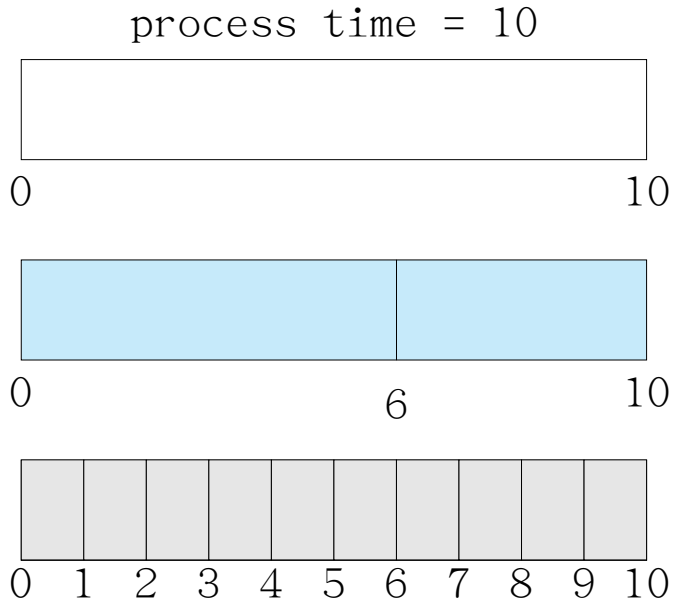
- Typically, **higher average turnaround** than SJF, but **better response**

Round Robin（时间片轮转，RR）Scheduling

- Performance
  - 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.
    - Example: 5 processes, time quantum=20ms
  - The performance of RR dependes heavily on the size of the time quantum.
    - if q is too large? ⇒FIFO
    - if q is too small? ⇒q must be large with respect to context switch, otherwise overhead is too high

Time Quantum and Context Switch Time

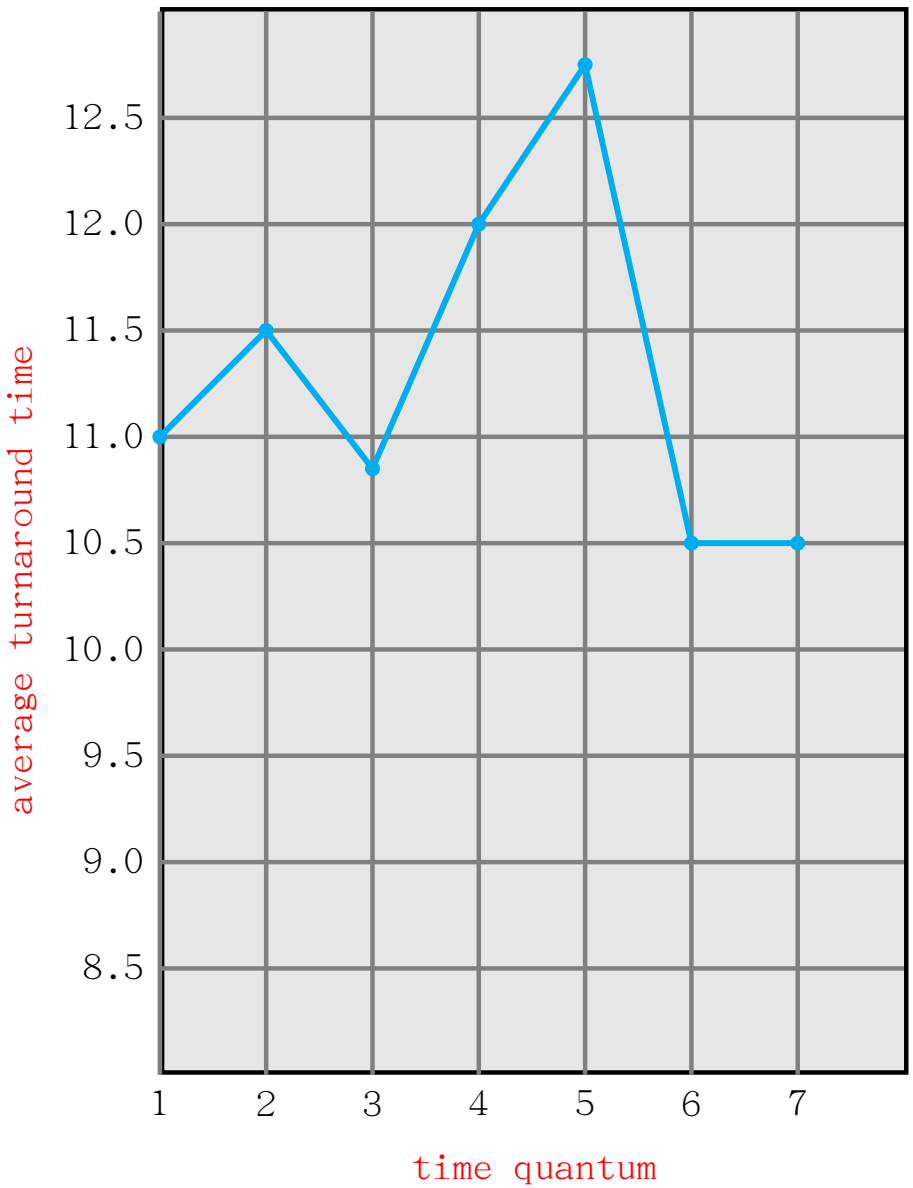
- The effect of context switching on the performance of RR scheduling



quantum	context switches
12	0
6	1
1	9

- typically the context-switch time is a small fraction of the time quantum
  - usually: time quantum: 10 ~100ms & context switch time: 10μs

Turnaround Time Varies With The Time Quantum



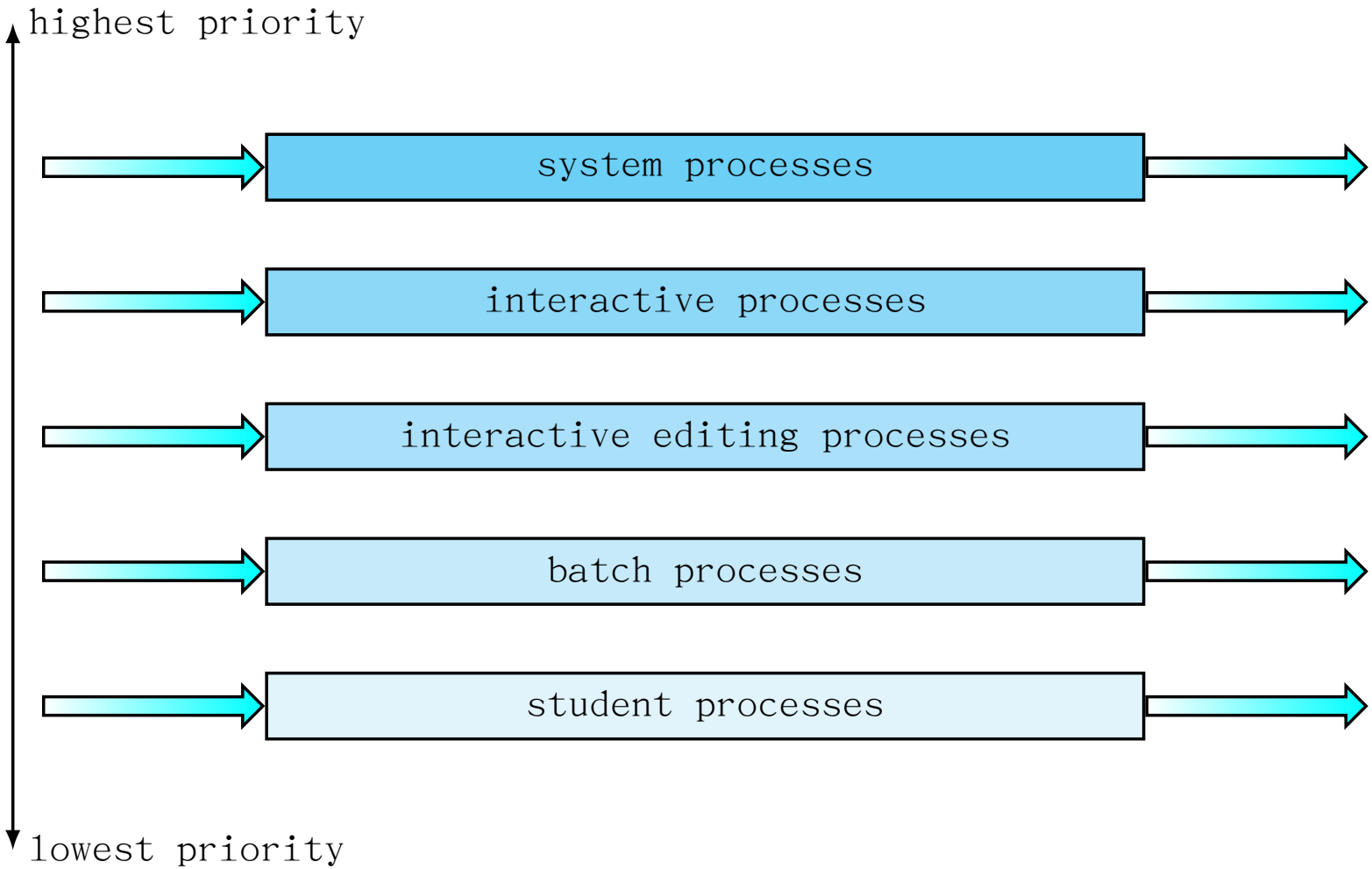
process	time
P_1	6
P_2	3
P_3	1
P_4	7

Outline

Multilevel Queue（多级队列） Scheduling I

- Ready queue is partitioned into separate queues. Each queue has its own scheduling algorithm
  - foreground (interactive) — RR
  - background (batch) — FCFS
- Scheduling must be done between the queues
  - Fixed priority scheduling;
    - Example: 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

example


highest priority
system processes
interactive processes
interactive editing processes
batch processes
student processes
lowest priority

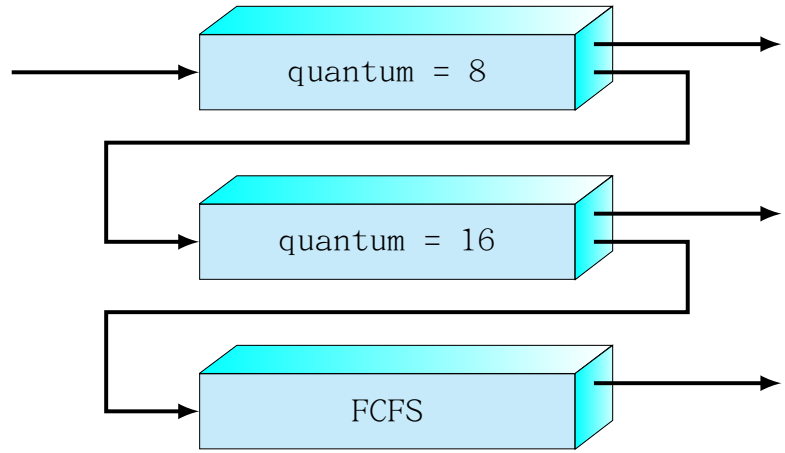
Outline

Multilevel-Feedback-Queue(多级反馈队列) Scheduling

- A process can **move** between the various queues; aging can be implemented this way
- Multilevel-feedback-queue(多级反馈队列) 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

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 — RR with time quantum 8ms
  - Q_1 — RR time quantum 16ms
  - Q_2 — FCFS



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

Outline

Multiple-Processor Scheduling

- One single processor $\rightarrow$ multiple CPUS
  - CPU scheduling more complex
  - Load sharing
- To be simple, suppose
  - the processors are identical — homogeneous — in terms of their functionality
  - so, any processor can execute any process in the queue

Multiple-Processor Scheduling

- Approches to Multiple-Processor Scheduling
  - **Asymmetric multiprocessing** — only one processor accesses the system data structures, alleviating the need for data sharing
  - **Symmetric multiprocessing** √
    - one common ready queue, or
    - one private ready queue for each processor

Multiple-Processor Scheduling

- **Processor Affinity**
  - Migration of processes from one processor to another processor **COST**s much.
    - For example: cache
    - most SMP systems try to avoid such migration
  - **Processor affinity**(亲和性):
    a process has an affinity for the processor on which it is currently running.
  - **SOFT** affinity VS. **HARD** affinity.

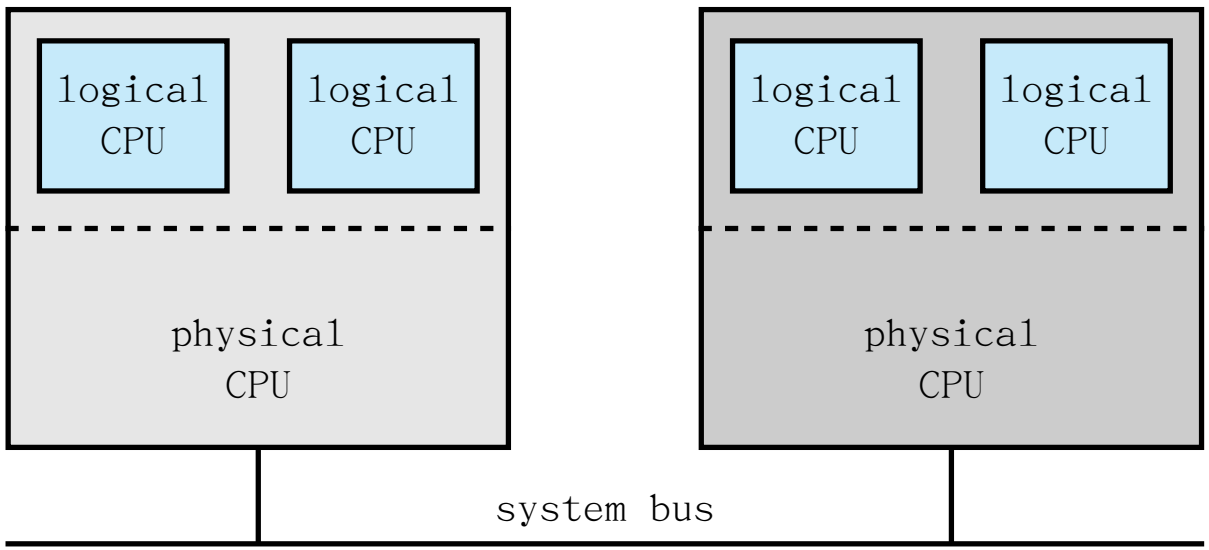
Multiple-Processor Scheduling

- Load Balancing
  - Load balanceing attempts to keep the workload evenly
    - for SMP system with one private ready queue for each processor
  - two general approaches
    - push migration(迁移)
    - pull migration

    often works together in load balancing systems
  - load balancing VS. processor affinity

Multiple-Processor Scheduling

- **Symmetric Multithreading**
  - INTEL: **hyperthreading technology (HT)**
  - **logical processors** VS. physical processors
    - each logical processor has its **own architecture state**, including general-purpose registers and machine-state registers, and interrupts
    - share: cache memory and bueses
  - ? from the viewpoint of OS ?

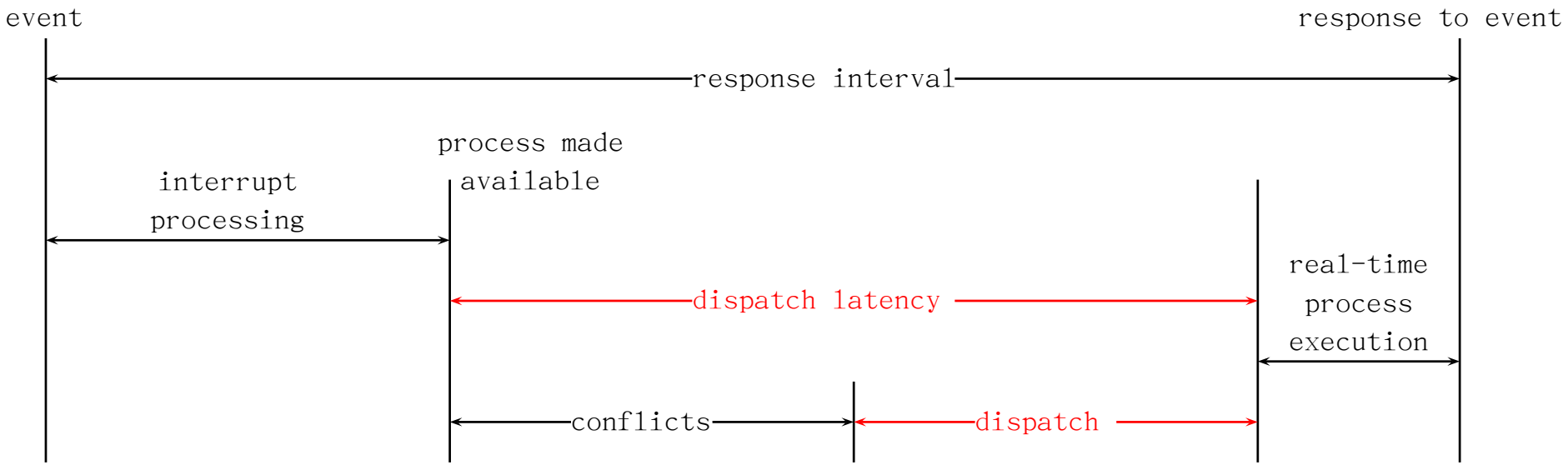


Outline

Real-Time Scheduling

- **Hard real-time systems** — required to complete a critical task within a guaranteed amount of time
- **Soft real-time computing** — requires that critical processes receive priority over less fortunate ones
- OS
  1. priority scheduling
  2. **short dispatch latency**
- approaches for short dispatch latency
  1. preemption
     - 1.1 preemption point (抢占点) in system calls with long period
     - 1.2 preemptible kernel
  2. priority inversion
     - 2.1 priority-inheritance protocol
     - 2.2 priority-ceiling protocol

dispatch latency



conflicts=preemption + resource releasing by processes with lower priority

Outline

OS examples

- READING
  - Solaris (thread)
  - Windows (thread)
  - Linux (process) $\surd$
  - μC/OS – II $\surd$

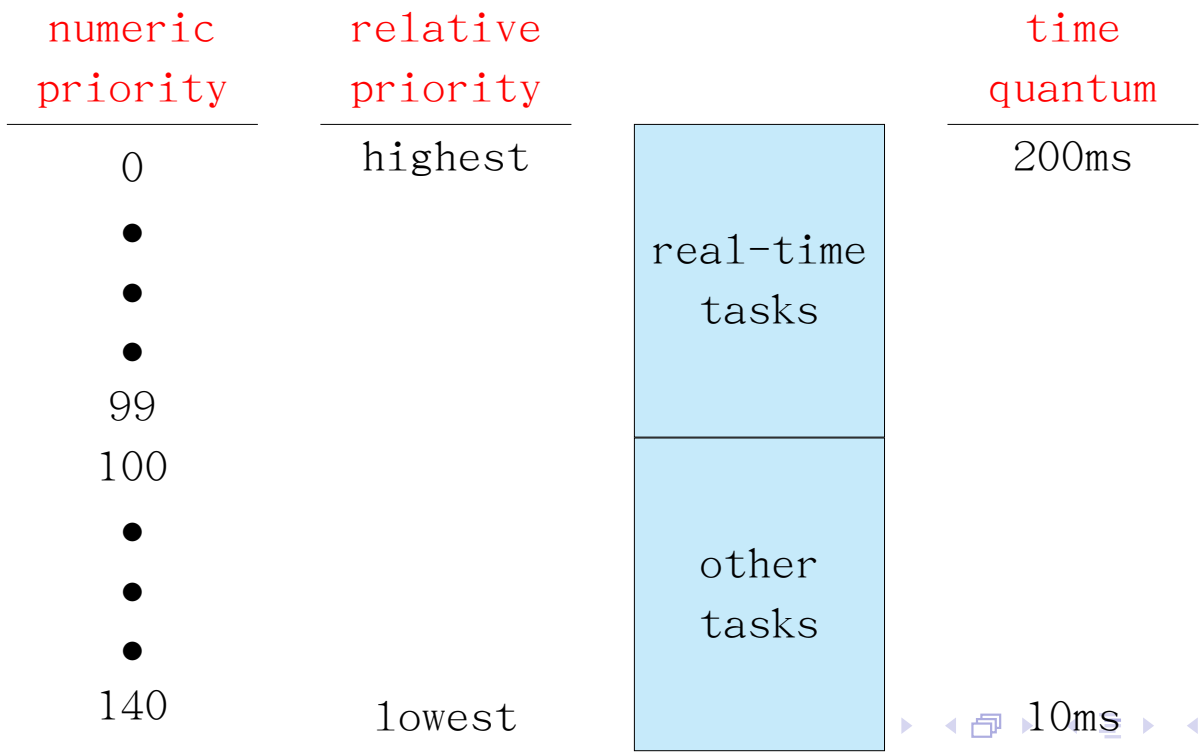
Outline

Linux Scheduling

- Linux is a general-purpose OS
  - Processes: time-sharing/real-time
  - Linux scheduler is both time-sharing-based and priority-based
    - With the changing of version, time-sharing technique changes too
- **Scheduling policy**:
  是一组规则，它们决定什么时候以怎样的方式选择一个新进程运行。
  Linux 2.6.26中
  - SCHED_NORMAL
  - SCHED_FIFO (for real-time process)
  - SCHED_RR (for real-time process)
  - SCHED_BATCH
  - SCHED_IDLE

Priorities

- The Linux scheduler: preemptive, priority-based
  - two seperate priority ranges: lower value $\equiv$ higher priority
    real-time range: 0~99
    a nice value rang: 100~140
- higher-priority $\Rightarrow$ longer time quanta
  (Unlike Solaris and Windows XP)

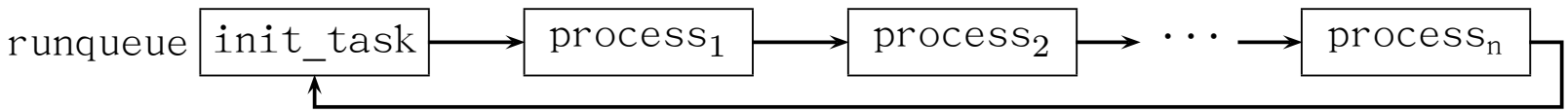


Priorities

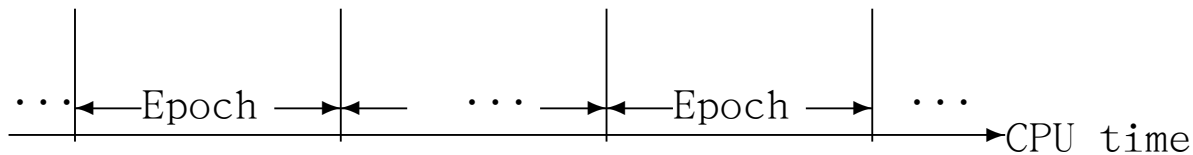
- The Linux scheduler: preemptive, priority-based
  - two seperate priority ranges: lower value $\equiv$ higher priority
    real-time range: 0~99
    a nice value rang: 100~140
- higher-priority $\Rightarrow$ longer time quanta
  (Unlike Solaris and Windows XP)
- Dynamic priorities:
  scheduler may change the priority of a process
  - 较长时间未分配到CPU的进程，通常↑
  - 已经在CPU上运行了较长时间的进程，通常↓

Linux scheduling algorithms

- Linux 2.4 scheduler
  - need to traverse the runqueue, O(n)

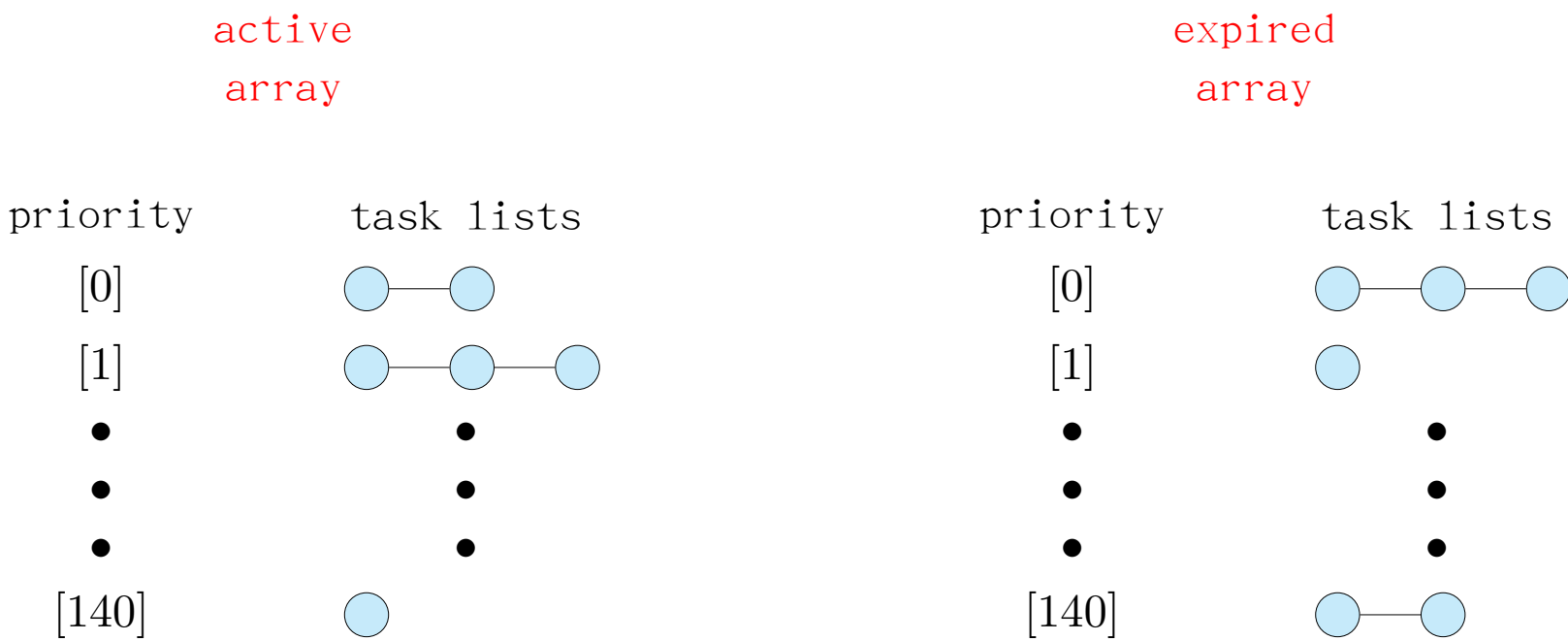


  - Epoch, default time slice (基本时间片), dynamic priorities



Linux scheduling algorithms

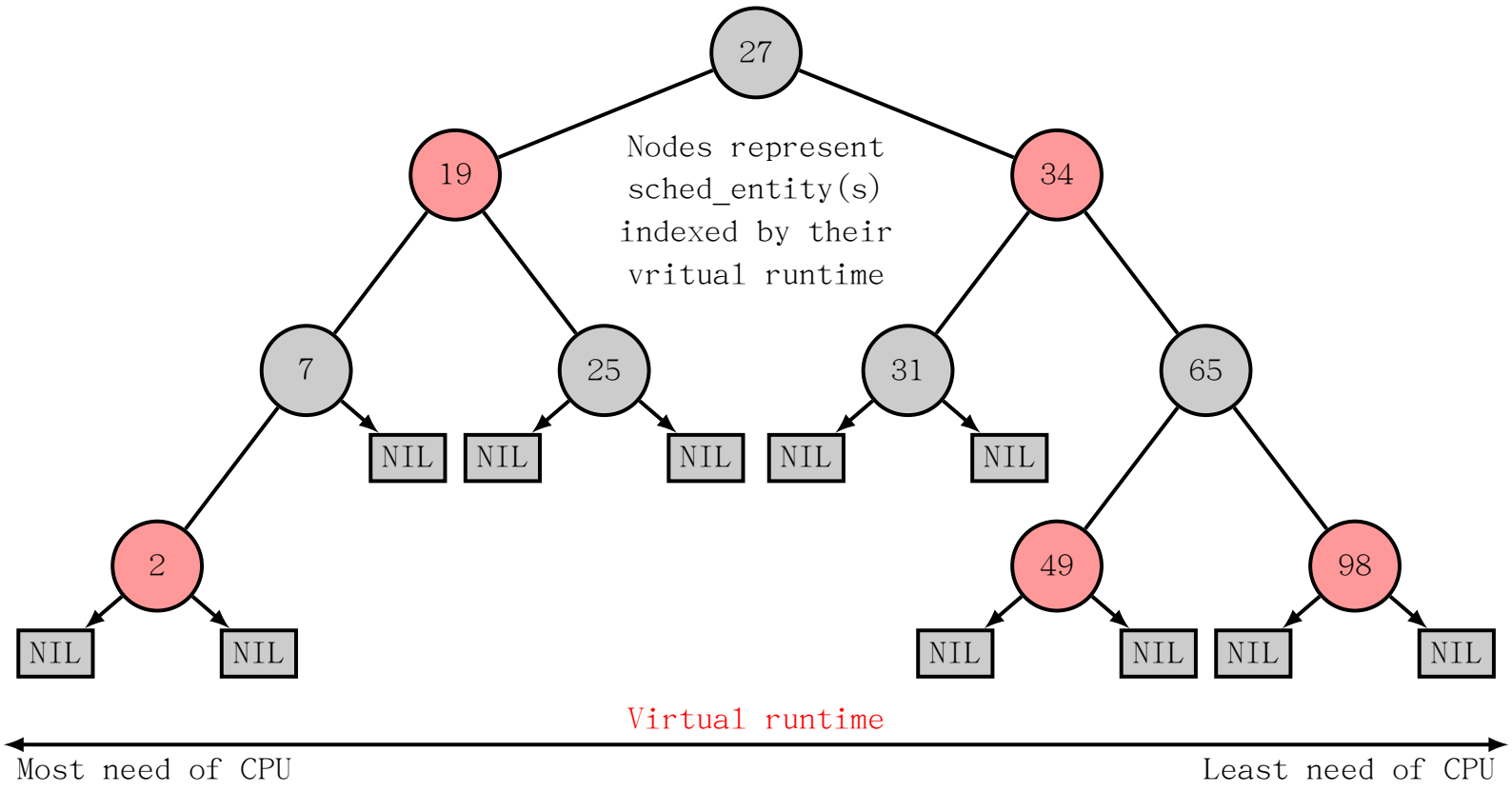
- Linux 2.6.17 scheduler (<2.6.23)
  - O(1)
  - Double priority-based arrays (双队列): active & expire



List of tasks indexed according to priorities

Linux scheduling algorithms

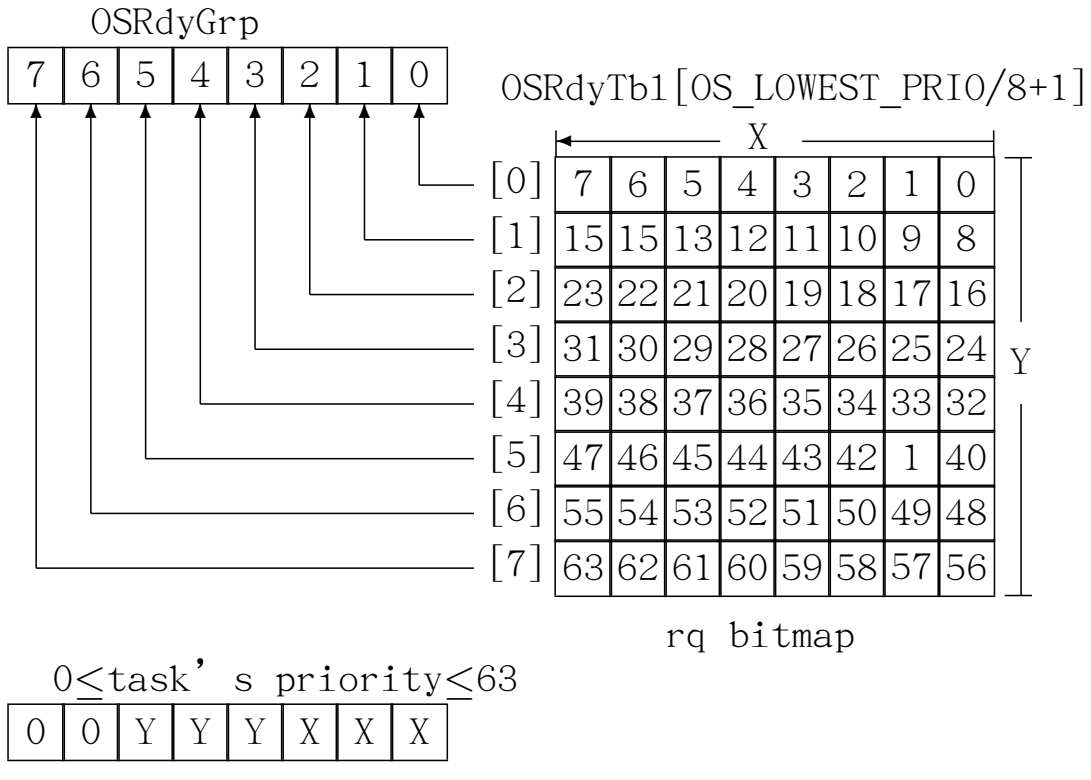
- Linux 2.6.26 scheduler (≥2.6.23)
  - O(1)
  - non-real-time: Complete-Fair-Scheduling(CFS, 完全公平调度), vruntime, red-black tree (红黑树)
  - real-time: priority arrays



Outline

uC/os-II scheduling

- Priority-based scheduler
  - MAX Tasks: 64
  - priority number: 0~63



Outline

Algorithm Evaluation

- How do we select a CPU scheduling algorithm for a particular system?
  - firstly, which criteria? What is the relative importance of these measures
  - then, evaluate the algorithms
    1. Deterministic Modeling(确定性建模)
    2. Queueing Models(排队模型)
    3. Simulations(模拟)
    4. Implementation

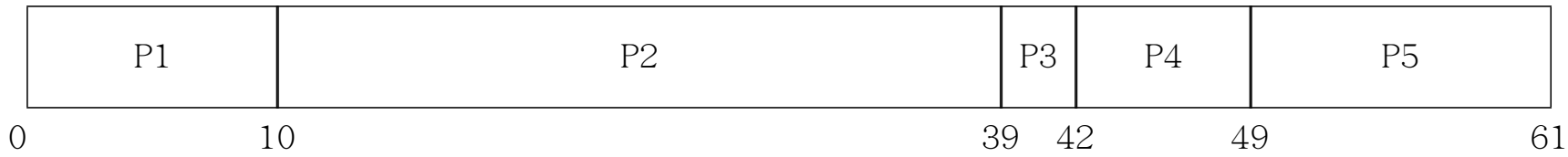
1. Deterministic Modeling(确定性建模) I

- **Analytic evaluation(分析评估法)**: One major class of evaluation methods
  - uses the given algorithm and the system workload to produce a formula or number that evaluates the performance of the algorithm for that workload.
- **Deterministic modeling(确定性建模)** — takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Example - Consider **FCFS**, **SJF**, and **RR** (quantum=10ms)

Process	BurstTime
P1	10
P2	29
P3	3
P4	7
P5	12

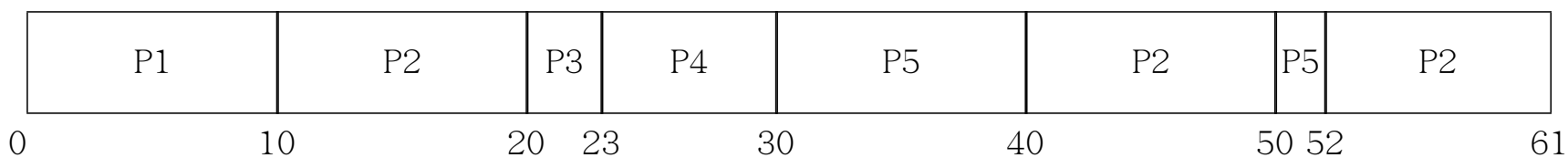
1. Deterministic Modeling(确定性建模) II

1. FCFS: average waiting time =(0+10+39+42+49)/5=28



2. SJF: average waiting time =(10+32+0+3+20)/5=13

3. RR: average waiting time
=(0+(10+20+2)+20+23+(30+10))/5=23



1. Deterministic Modeling(确定性建模) III

- advantages and disadvantages
  - 确定性 vs. 适用性和实用性

2. Queueing Models(排队模型)

- Usually, two distributions can be measured and then approximated or simply estimated
  - the distribution of CPU and I/O bursts
  - the arrival-time distribution
- Queueing-network analysis(排队网络分析)
  - Computer System: a network of servers, each server has a queue of waiting processes
    - CPU: ready queue;
    - I/O: device queues (≡waiting queue)
  - Given arriving rates and service rates ⇒utilization, average queue length, average wait time, ...

2. Queueing Models(排队模型)

- Example:

  1. n: the average queue length
  2. W: the average waiting time
  3. λ: the average arrival rate

  for a steady waits (Little formula, Little公式):

$$n = \lambda \times W$$

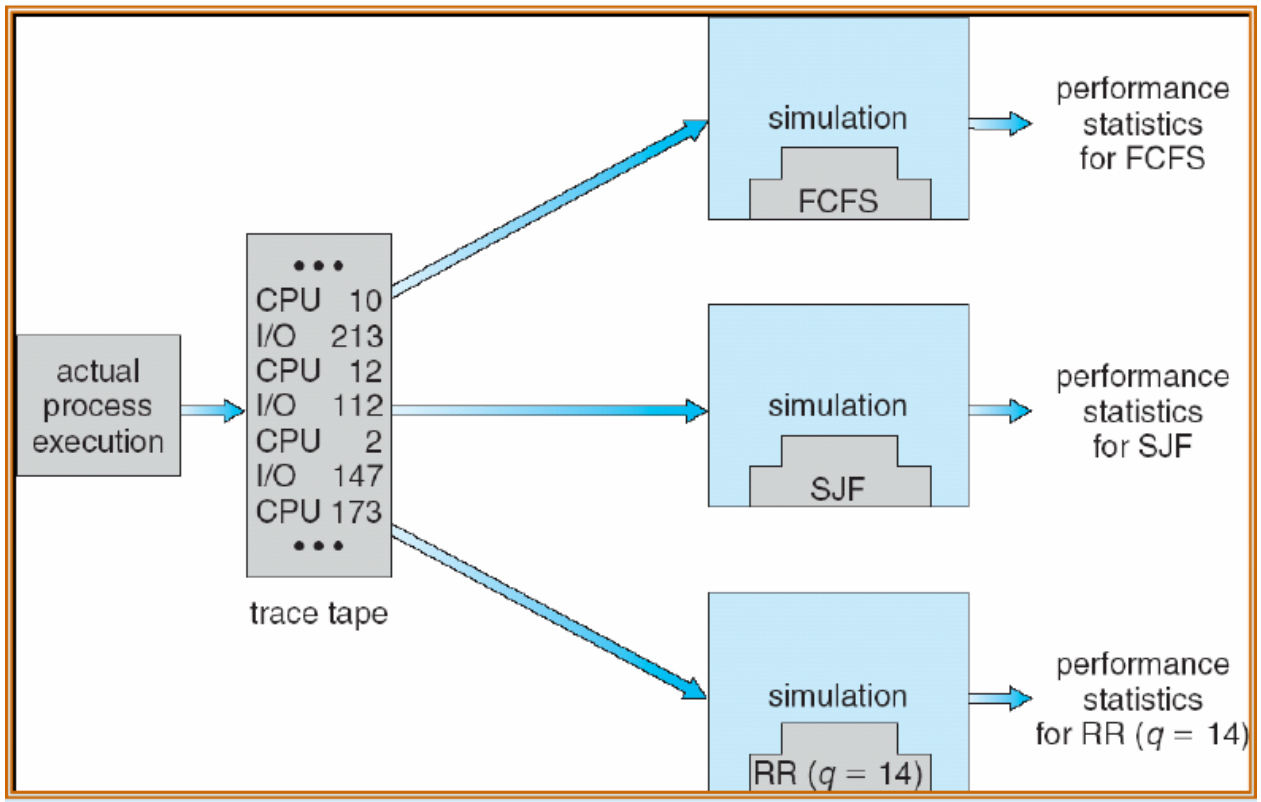
  - Little formula is particularly useful because it is valid for any scheduling algorithm and arrival distribution.
  - If we know two of the three variables, we can use Little formula to compute the other one.

3. Simulations(模拟) I

- Running simulations involves **programming a model of the computer system**.
  - Software data structures represent the major components
    - a clock
    - the system state is modified to reflect the activities of the devices, the processes and the scheduler.
  - finally, the statictics are gathered
- How to generate the data to drive the simulation?
  - **distribution-driven simulation**
    - **ramdon-number generator**, according to probability distributions, to generate processes, CPU burst times, arrivals, departures, ...
    - the distributions can be defined mathematically(uniform, exponential, Poisson) or empirically
    - may be inaccurate

3. Simulations(模拟) II

- trace tapes(跟踪磁带)



eveluation of CPU schedulers by simulation

4. Implementation

- This approach put the actual algorithm in the real system for evaluation under real operating conditions
- the main difficulty: high cost

Outline

小结

作业 I

- 6.1 一个CPU调度算法决定了它所调度的进程的执行顺序。如果在一个处理器上有n个进程要被调度，可能有多少种不同的调度算法？给出一个用n表示的公式。
- 6.3 考虑下列进程集，进程占用的CPU区间时间长度以毫秒来计算：

进程	区间时间	优先级
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

假设在时刻0进程以P_1，P_2，P_3，P_4，P_5的顺序到达。

  - a．画出4个Gantt图分别演示用FCFS、SJF、非抢占优先级（数字小代表优先级高）和RR（时间片=1）算法调度时进程的执行过程。
  - b．在a里每个进程在每种调度算法下的周转时间是多少？

作业 II

- c．在a里每个进程在每种调度算法下的等待时间是多少?
- d．在a里结果哪一种调度算法的平均等待时间对所有进程而言最小?

- 6.4 假设下列进程在所指定的时刻到达等待执行。每个进程将运行所列出的时间量长度。在回答问题时，假设使用非抢占式调度算法，基于选择时你所拥有的信息作出决定。

进程	到达时间	区间时间
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

  - a．当使用FCFS调度算法时，这些进程的平均周转时间是多少?
  - b．当使用SJF调度算法时，这些进程的平均周转时间是多少?

作业 III

- c．SJF调度算法被认为能提高性能，但是注意在时刻0选择运行进程P1因为无法知道两个更短的进程很快会到来。计算一下如果在第一个时间单元CPU被置为空闲，然后使用SJF调度算法，计算这时的平均周转时间是多少？注意在空闲时，进程P1和P2在等待，所以他们的等待时间可能会增加。这个算法刻意被认为是预知（future-knowledge）调度。

- 6.7 考虑下面的基于动态改变优先级的可抢占式优先权调度算法。大的优先权数代表高优先权。当一个进程在等待CPU时（在就绪队列中，但未执行），优先权以α 速率改变；当它运行时，优先权以β速率改变。所有的进程在进入就绪队列时被给定优先权为0。参数α和β可以设定给许多不同的调度算法。
  - a．$\beta > \alpha > 0$ 时所得的是什么算法?
  - b．$\alpha < \beta < 0$ 时所得的是什么算法?

作业 IV

- 6.8 许多CPU调度算法可以设置参数。例如，RR算法需要一个参数来指定时间片。多级反馈队列需要一些参数来定义队列的数，每一个队列的调度算法，在队列之间移动进程的标准，等等。 这些算法就成了一个算法集合（例如所有时间片的RR算法集合等）。一个算法集合可以包括另一个（例如FCFS算法是一个时间片无限的RR算法）。下列各对算法集之间是否有联系，如果有是什么?
  - a. 优先级和SJF
  - b. 多层反馈队列和FCFS
  - c. 优先级和FCFS
  - d. RR和SJF

谢谢！