# 0117401: Operating System
# 操作系统原理与设计
## 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

March 27, 2024

# 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

# Outline

1. Basic Concepts
   - CPU-I/O Burst Cycle
   - CPU Scheduler
   - Preemptive Scheduling
   - Dispatcher

# Basic Concepts

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

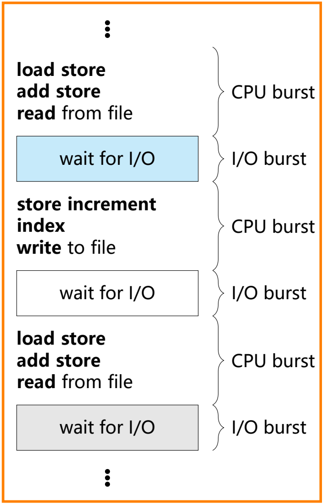
## Objective of multiprogramming

- **Maximum CPU utilization**

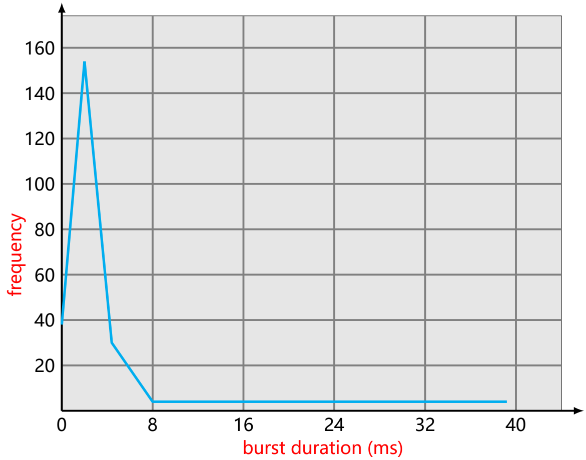
# Outline

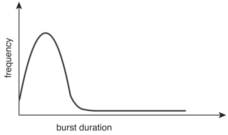
# Basic Concepts: CPU-I/O Burst Cycle

- An observed **property** of process: **Process execution consists of a cycle of CPU execution & I/O wait**
- Alternating Sequence of CPU & I/ O Bursts
- Begin & end with a CPU burst
- Process execution
  = n (CPU execution + I/O wait)
  + CPU execution



# CPU burst distribution




(10th, Fig5.2)

Histogram of CPU-burst Times

# Outline

1. Basic Concepts
   - CPU-I/O Burst Cycle
   - CPU Scheduler
   - Preemptive Scheduling
   - Dispatcher

# CPU Scheduler

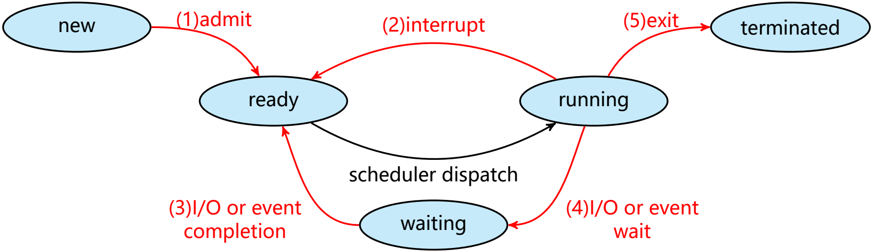
- **CPU scheduler** (Short-term Scheduler) **selects** a process from the processes in memory that are ready to execute and **allocates** the CPU to the process
- **Ready Queue** could be:
  - a FIFO Queue?
  - a priority queue?
  - a tree?
  - an unordered linked list?

# 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
   - 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

# Preemptive Scheduling III

## 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**?

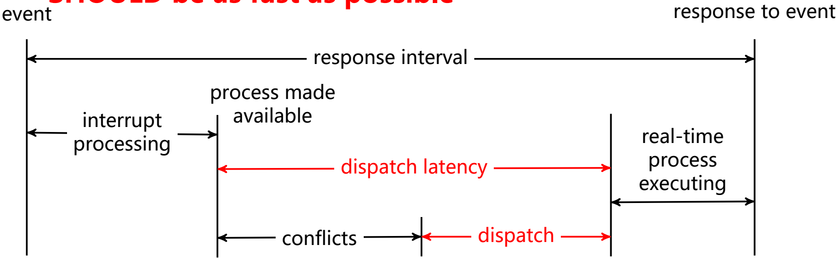
- new mechnisms are needed, such as
  - **disable interrupt**
  - some **synchronization mechnisms**

# Outline

1. Basic Concepts
   - CPU-I/O Burst Cycle
   - CPU Scheduler
   - Preemptive Scheduling
   - Dispatcher

# 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

2 Scheduling Criteria

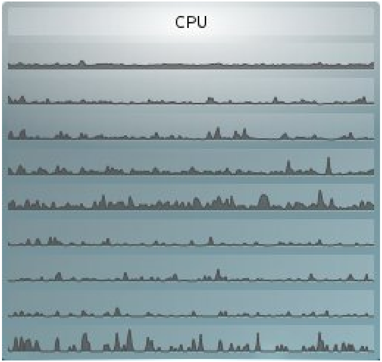
- Scheduling Criteria

# 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)
5. **Response time** (响应时间)

# 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

3 Scheduling Algorithms
- FCFS(先来先服务) Scheduling
- SJF(短作业优先) Scheduling
- Priority Scheduling
- Round Robin(时间片轮转) Scheduling
- Multilevel Queue（多级队列） Scheduling
- Multilevel Feedback Queue（多级反馈队列） Scheduling

# Outline

3. Scheduling Algorithms
   - FCFS(先来先服务) Scheduling
   - SJF(短作业优先) Scheduling
   - Priority Scheduling
   - Round Robin(时间片轮转) Scheduling
   - Multilevel Queue（多级队列） Scheduling
   - Multilevel Feedback Queue（多级反馈队列） Scheduling

# 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

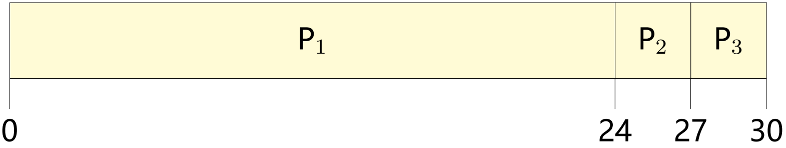
# Example of FCFS Scheduling

- 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

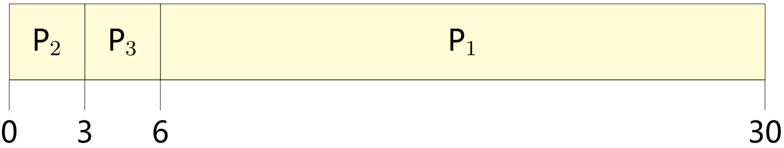
# Example of FCFS Scheduling II

- 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
- $\equiv$short process behind long process

## example situation:

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

# Outline

3. Scheduling Algorithms
   - FCFS(先来先服务) Scheduling
   - SJF(短作业优先) Scheduling
   - Priority Scheduling
   - Round Robin(时间片轮转) Scheduling
   - Multilevel Queue（多级队列） Scheduling
   - Multilevel Feedback Queue（多级反馈队列） Scheduling

# SJF Scheduling

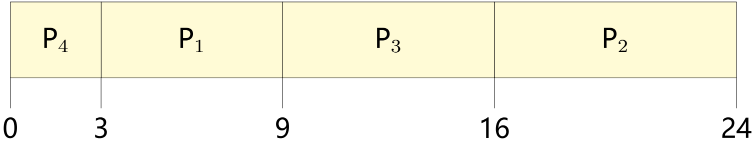
- **Shortest-Job-First(短作业优先)**
  **Shortest-Next-CPU-Burst algorithm**
  - Associate with each process **the length of its next CPU burst**.
  - **Schedule the process with the shortest time**.

# 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

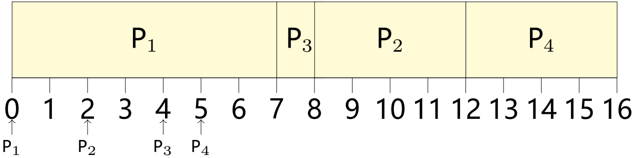
- Two schemes:
  1. **nonpreemptive**
     – once CPU given to the process it cannot be preempted until completes its CPU burst
  2. **preemptive**
     – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the **Shortest-Remaining-Time-First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



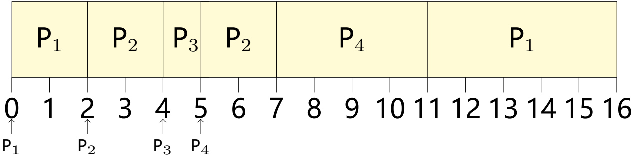

- **Average waiting time**: $(0+6+3+7)/4=4$

# 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

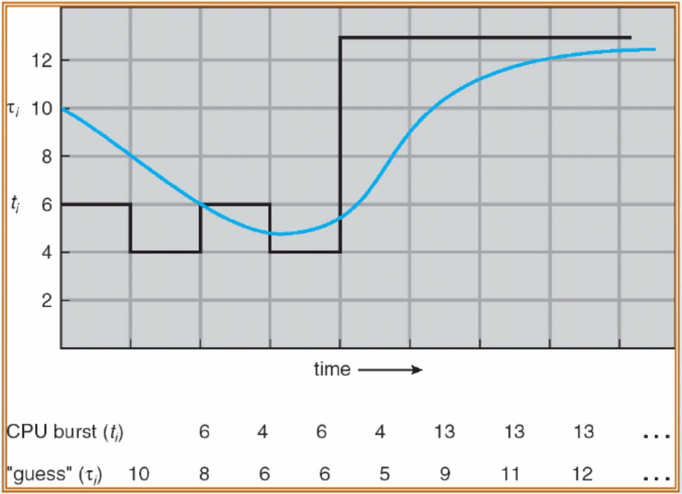
- Example: by using the length of previous CPU bursts, using **exponential averaging(指数平均)**
  1. $t_n$ =actual length of $n^{th}$ CPU burst
  2. $\tau_{n+1}$= predicted value for the next CPU burst
  3. $\alpha, 0 \leq \alpha \leq 1$
  4. Define: $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$
- If we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

3. Scheduling Algorithms
   - FCFS(先来先服务) Scheduling
   - SJF(短作业优先) Scheduling
   - Priority Scheduling
   - Round Robin(时间片轮转) Scheduling
   - Multilevel Queue（多级队列） Scheduling
   - Multilevel Feedback Queue（多级反馈队列） Scheduling

# Priority(优先级) Scheduling

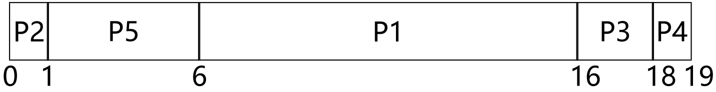
- A **priority number(优先数)** is associated with each process
  - **priority number(优先数)** VS. **priority(优先级)**
  - usually an integer, &
    usually, **smallest integer** $\equiv$ **highest priority**
- The CPU is allocated to the process with the highest priority
  - **Preemptive** VS. **Nonpreemptive**
- **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

3 Scheduling Algorithms
- FCFS(先来先服务) Scheduling
- SJF(短作业优先) Scheduling
- Priority Scheduling
- Round Robin(时间片轮转) Scheduling
- Multilevel Queue（多级队列） Scheduling
- Multilevel Feedback Queue（多级反馈队列） Scheduling

# Round Robin (时间片轮转，RR) Scheduling

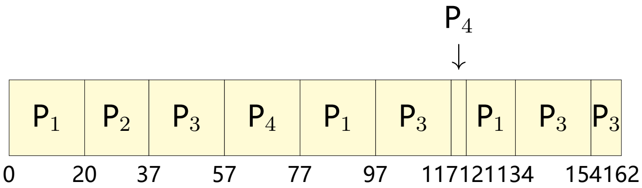
- **Time quantum, time slice(时间片)**
  - a small unit of CPU time
  - usually 10-100 ms
- **Implementation**
  - **Ready queue**: a FIFO circular queue
  - Each process gets 1 time quantum
  - **Insert**: to the tail of the queue
  - **Scheduling**: pick the first process; set timer; and dispatch
  - **two situation**:
    - ★ CPU burst $\leq 1$ 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:



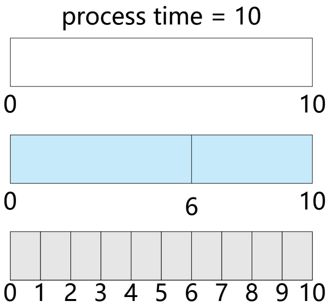

- Typically, **higher average turnaround** than SJF, but **better response**

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

# Time Quantum and Context Switch Time

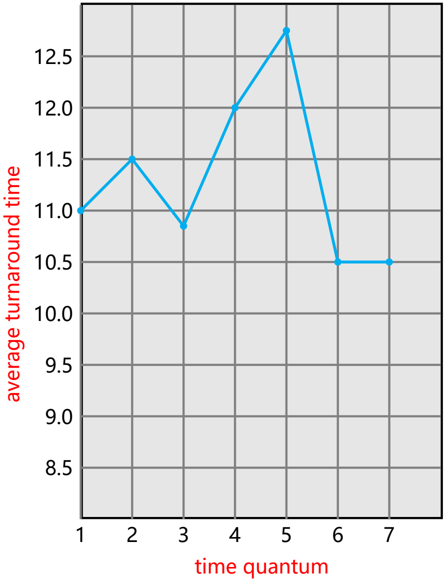
- **The effect of context switching on the performance of RR scheduling**




| 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: 10$\mu$s

# Turnaround Time Varies With The Time Quantum




| process | time |
|---|---|
| $P_1$ | 6 |
| $P_2$ | 3 |
| $P_3$ | 1 |
| $P_4$ | 7 |

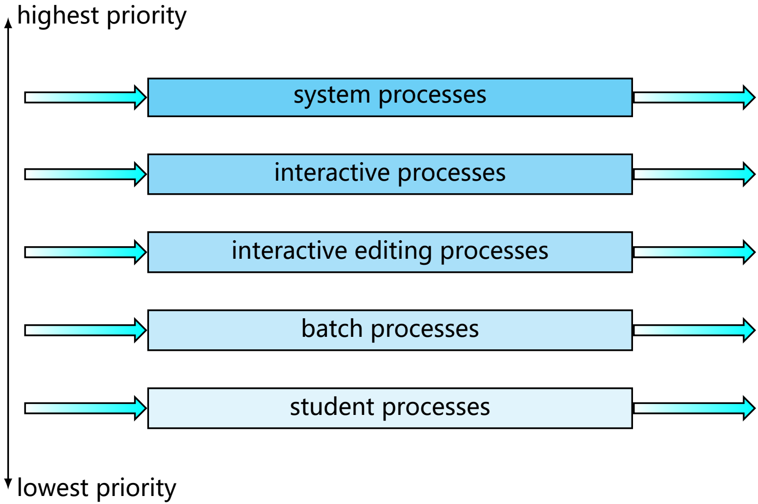
# Outline

3. Scheduling Algorithms
   - FCFS(先来先服务) Scheduling
   - SJF(短作业优先) Scheduling
   - Priority Scheduling
   - Round Robin(时间片轮转) Scheduling
   - Multilevel Queue（多级队列） Scheduling
   - Multilevel Feedback Queue（多级反馈队列） Scheduling

# 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

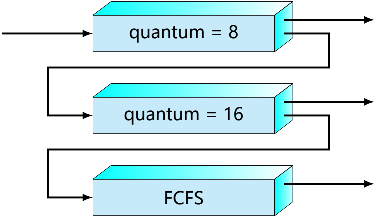
3. Scheduling Algorithms
   - FCFS(先来先服务) Scheduling
   - SJF(短作业优先) Scheduling
   - Priority Scheduling
   - Round Robin(时间片轮转) Scheduling
   - Multilevel Queue（多级队列） Scheduling
   - Multilevel Feedback Queue（多级反馈队列） Scheduling

# 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** $\surd$
    - 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.

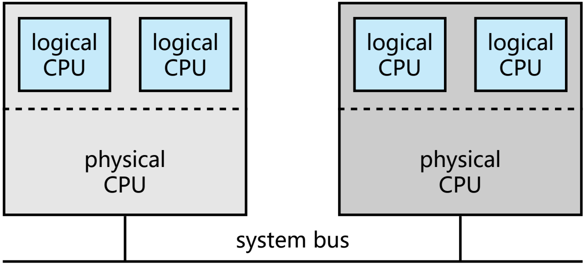
# Multiple-Processor Scheduling

- 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 ?

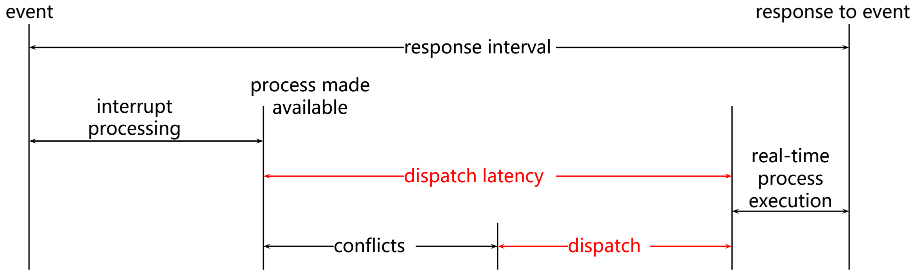


# Outline

# 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. preemption point (抢占点) in system calls with long period
     2. preemptible kernel
  2. priority inversion
     1. priority-inheritance protocol
     2. priority-ceiling protocol

# dispatch latency




conflicts=preemption + resource releasing by processes with lower priority

# Outline

6 OS examples
- Linux Scheduling
- uC/os-II scheduling

# OS examples

- READING
  - Solaris (thread)
  - Windows (thread)
  - Linux (process)$\surd$
  - $\mu$C/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)

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

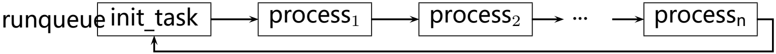
The Relationship Between Priorities and Time-slice length List of Tasks

# Priorities

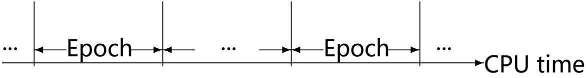
- The **Linux scheduler**: **preemptive**, **priority-based**
  - two seperate priority ranges: lower value $\equiv$ higher priority
    real-time range: 0~99
    a nice value rang: 100~140
- **higher**-priority $\Rightarrow$ **longer** time quanta
  (Unlike Solaris and Windows XP)
- **Dynamic priorities**: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



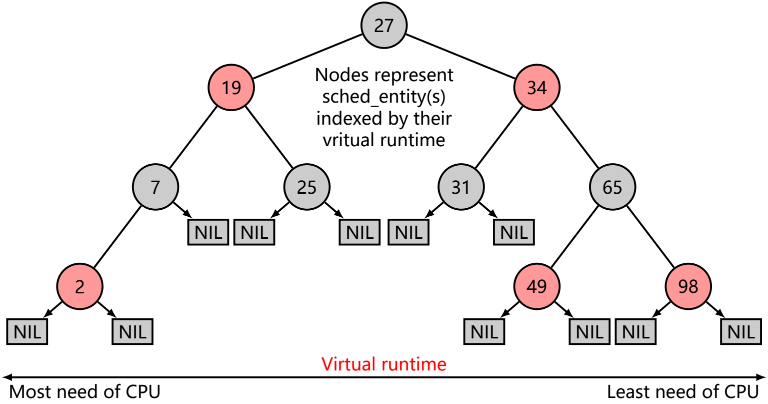
# Linux scheduling algorithms

- 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

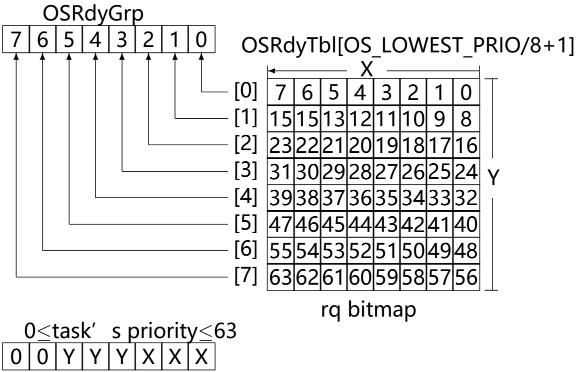


# Outline

6. OS examples
   - Linux Scheduling
   - uC/os-II scheduling

# uC/os-II scheduling

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



# Outline

7 Algorithm Evaluation

# 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

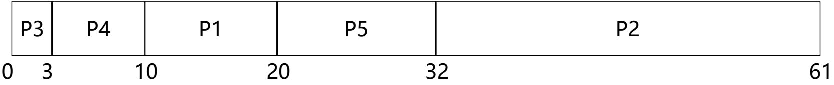
- **Analytic evaluation(分析评估法)**: One major class of evaluation methods
  - **uses the given algorithm and the system workload to produce a formula or number that evaluates the performance of the algorithm for that workload.**
- **Deterministic modeling(确定性建模)** – takes a particular predetermined workload 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 |

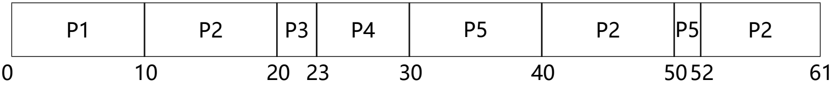
# 1. Deterministic Modeling(**确定性建模**) II

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




- **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. $\lambda$: the average arrival rate

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

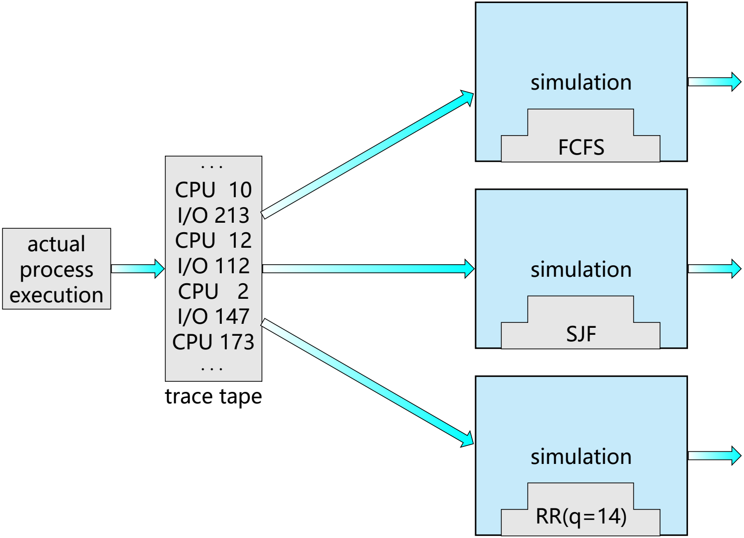
$$n = \lambda \times W$$

  - 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
  - **trace tapes(跟踪磁带)**

# 3. Simulations(模拟) II




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

8 小结

# 小结

1. Basic Concepts
   - CPU-I/O Burst Cycle
   - CPU Scheduler
   - Preemptive Scheduling
   - Dispatcher
2. Scheduling Criteria
   - Scheduling Criteria
3. Scheduling Algorithms
   - FCFS(先来先服务) Scheduling
   - SJF(短作业优先) Scheduling
   - Priority Scheduling
   - Round Robin(时间片轮转) Scheduling
   - Multilevel Queue（多级队列） Scheduling
   - Multilevel Feedback Queue（多级反馈队列） Scheduling
4. Multiple-Processor Scheduling
5. Real-Time Scheduling
6. OS examples
   - Linux Scheduling
   - uC/os-II scheduling
7. Algorithm Evaluation
8. 小结

**Thank you! Any question?**