

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

March 22, 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 criteria** for selecting a CPU-scheduling algorithm for a particular system.

提纲——CPU scheduling

- 1 Basic Concepts
- 2 Scheduling Criteria
- 3 Scheduling Algorithms
- 4 Multiple-Processor Scheduling
- 5 Real-Time Scheduling
- 6 OS examples
- 7 Algorithm Evaluation
- 8 小结和作业

1 Basic Concepts

- CPU-I/O Burst Cycle
- CPU Scheduler
- Preemptive Scheduling
- Dispatcher

- 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

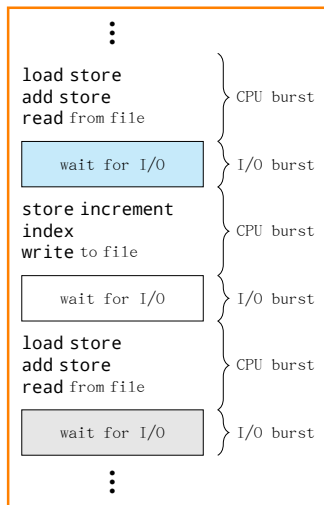
- Maximum CPU utilization

1 Basic Concepts

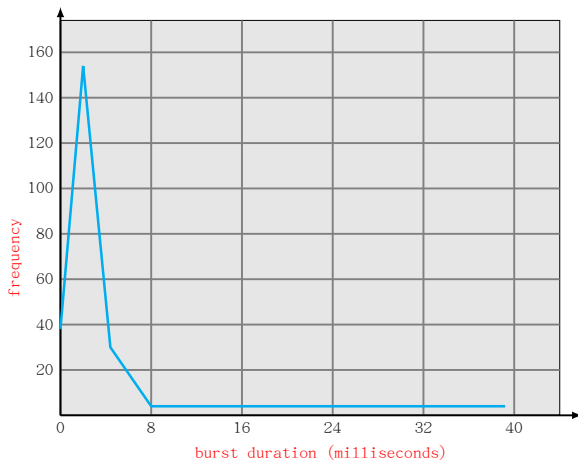
- CPU-I/O Burst Cycle
- CPU Scheduler
- Preemptive Scheduling
- Dispatcher

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**
 $= n (\text{CPU execution} + \text{I/O wait}) + \text{CPU execution}$



CPU burst distribution



Histogram of CPU-burst Times

1 Basic Concepts

- CPU-I/O Burst Cycle
- CPU Scheduler
- Preemptive Scheduling
- Dispatcher

- **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?

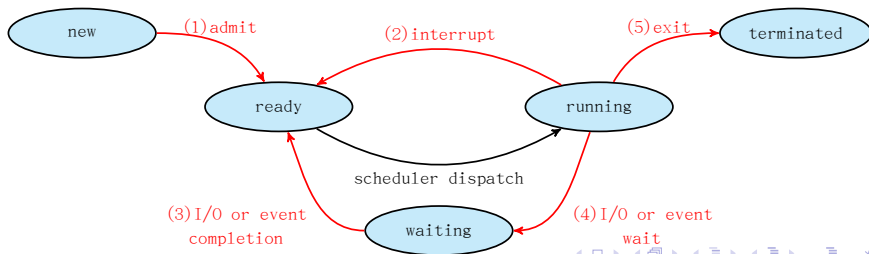
- 1 Basic Concepts
 - CPU-I/O Burst Cycle
 - CPU Scheduler
 - Preemptive Scheduling
 - Dispatcher

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?



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

COST for preemption

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

Preemption of the OS kernel

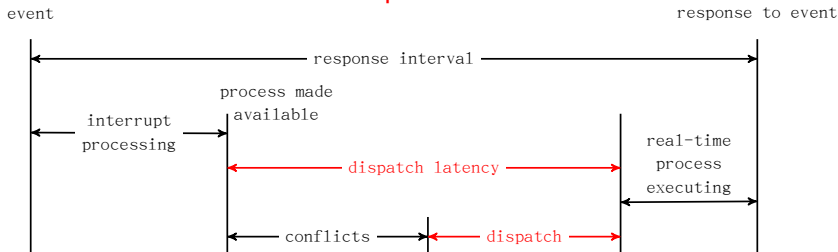
- What happens if the process is preempted in the middle of some activities that changes important **kernel data**?
- preemptive kernel VS. nonpreemptive kernel?
- Interrupt affected code VS normal kernel code?
- new mechanisms are needed, such as
 - **disable interrupt**
 - some **synchronization mechanisms**

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:
 - switching context
 - switching to user mode
 - 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**



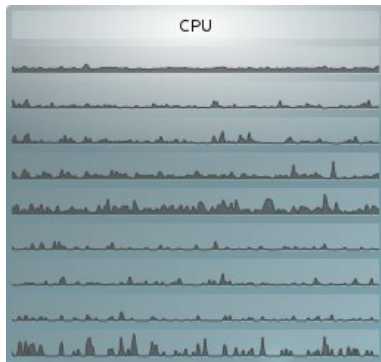
- 2 Scheduling Criteria
 - Scheduling Criteria

- 2 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% ~ 100%; in a real system: 40% ~ 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)

- 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)

- 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)

- 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)

- 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

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

different from system to system.

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

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

- **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

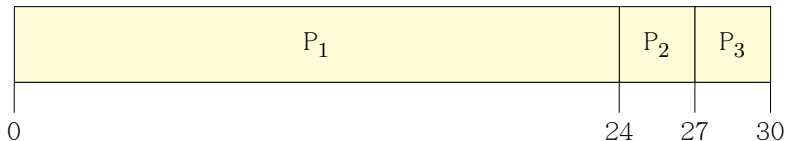
Example of FCFS Scheduling

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

P1 , P2 , P3

<u>Process</u>	<u>BurstTime(ms)</u>
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:** $(0 + 24 + 27)/3 = 17$

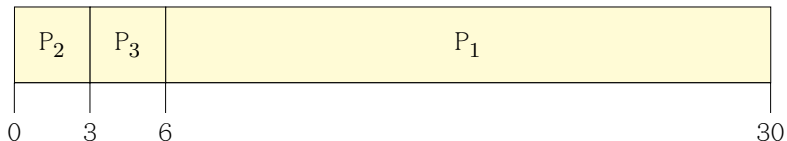
Example of FCFS Scheduling II

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

P2 , P3 , P1

<u>Process</u>	<u>BurstTime(ms)</u>
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

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

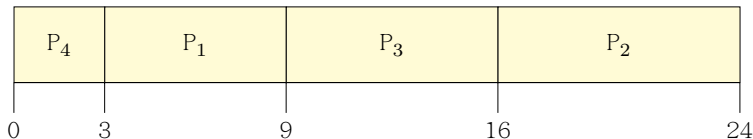
- Shortest-Job-First(短作业优先)
Shortest-Next-CPU-Burst algorithm
 - Associate with each process the length of its next CPU burst.
 - Schedule the process with the shortest time.

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 is **optimal**(最优的)

— gives **minimum average waiting time**(最小平均等待时间)
for a given set of processes

- Two schemes:

- ① **nonpreemptive**

- once CPU given to the process it cannot be preempted until completes its CPU burst

- ② **preemptive**

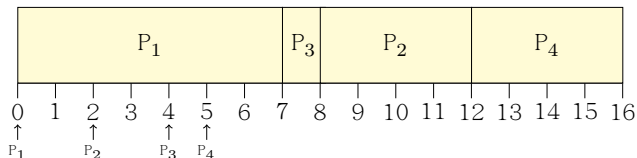
- if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the **Shortest-Remaining-Time-First (SRTF)**

SJF scheduling schemes

Example of Non-Preemptive SJF

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

The Gantt chart for SJF (non-preemptive)



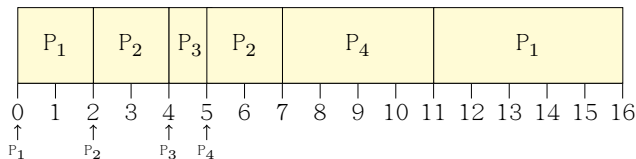
Average waiting time: $(0 + 6 + 3 + 7)/4 = 4$

SJF scheduling schemes

● Example of **Preemptive** SJF

<u>Process</u>	<u>ArrivalTime</u>	<u>BurstTime(ms)</u>
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: $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$

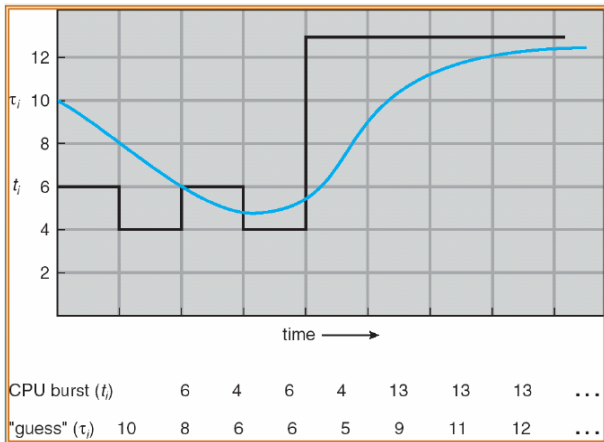
- If we expand the formula, we get:

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

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



- Examples of Exponential Averaging

- if $\alpha = 0$

- $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n = 0 \cdot t_n + \tau_n = \tau_n$
- Recent history does not count

- if $\alpha = 1$

- $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n = t_n + 0 \cdot \tau_n = t_n$
- Only the actual last CPU burst counts

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

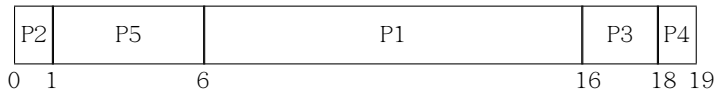
- A **priority number(优先数)** is associated with each process
 - **priority number(优先数)** VS. **priority(优先级)**
 - usually an integer, & usually, **smallest integer \equiv highest priority**
- The CPU is allocated to the process with the highest priority
 - **Preemptive** VS. **Nonpreemptive**
- **SJF 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:** $(6 + 0 + 16 + 18 + 1)/5 = 8.2$

- 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 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?

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

- 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.

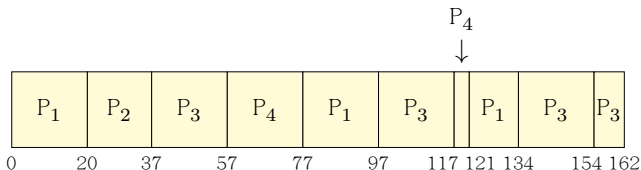
After this time has elapsed, the process is preempted(被抢占) and added to the end of the ready queue.

Round Robin (时间片轮转, RR) Scheduling

- Example of RR with Time Quantum = 20

<u>Process</u>	<u>BurstTime</u>
P1	53
P2	17
P3	68
P4	24

- The **Gantt chart** is:



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

- Performance

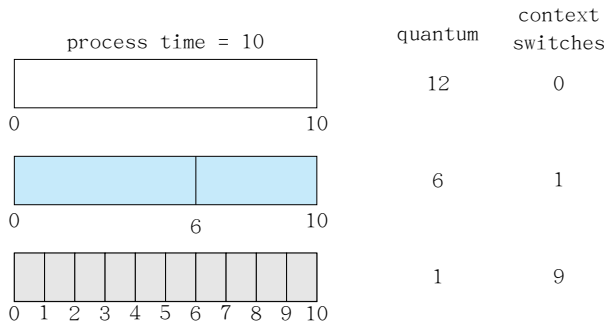
- If there are n processes in the ready queue and the time quantum is q , then **each process gets $1/n$ of the CPU time** in chunks of at most q time units at once.

No process waits more than $(n-1)q$ time units.

- Example: 5 processes, time quantum=20ms
- The **performance** of RR depends 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

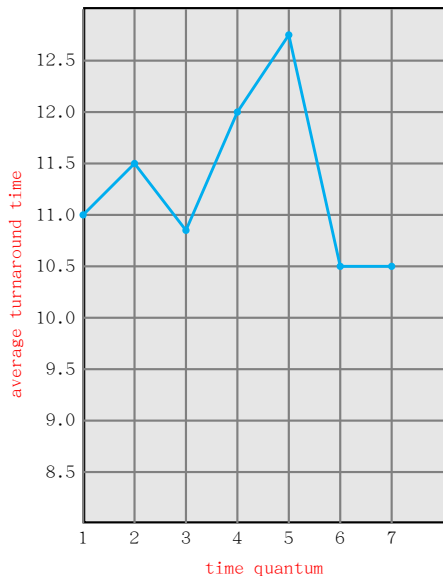
Time Quantum and Context Switch Time

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



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

Turnaround Time Varies With The Time Quantum



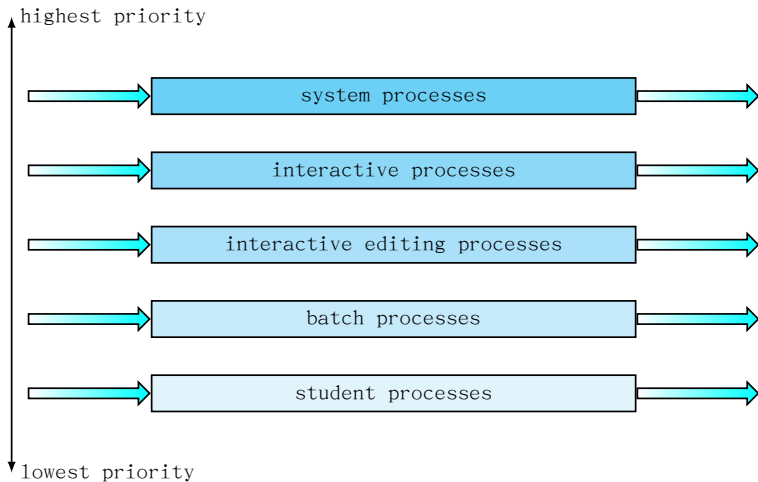
process	time
P ₁	6
P ₂	3
P ₃	1
P ₄	7

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

- 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



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

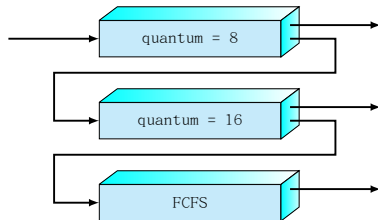
Multilevel-Feedback-Queue (多级反馈队列) Scheduling

- 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 .

4 Multiple-Processor Scheduling

Multiple-Processor Scheduling

- One single processor → 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

- 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

● 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.

- Load Balancing

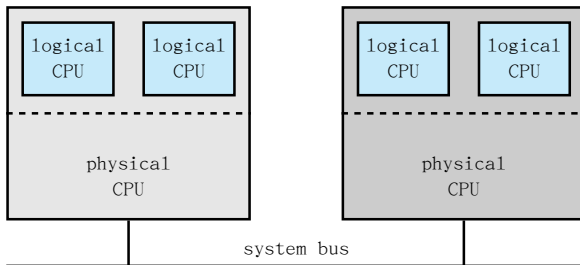
- Load balancing attempts to **keep the workload evenly**
 - for SMP system with one private ready queue for each processor
- two general approaches
 - **push migration(迁移)**
 - **pull migration**

often **works together** in load balancing systems

- **load balancing VS. processor affinity**

• 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 buses
- ? from the viewpoint of OS ?

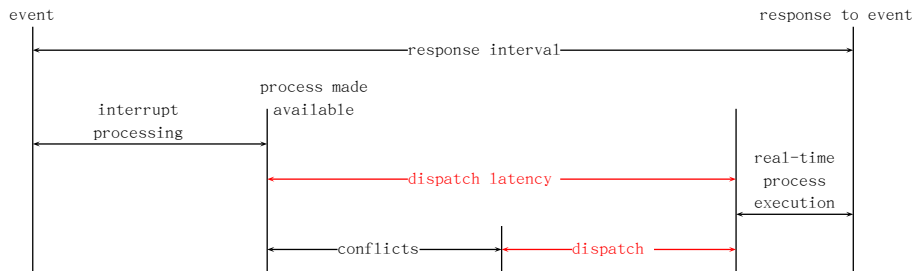


5 Real-Time Scheduling

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



$\text{conflicts} = \text{preemption} + \text{resource releasing by processes with lower priority}$

6 OS examples

- Linux Scheduling
- uC/os-II scheduling

- READING

- Solaris (thread)
- Windows (thread)
- Linux (process) ✓
- $\mu C/OS - II$ ✓

- 6 OS examples
 - Linux Scheduling
 - uC/os-II 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:**

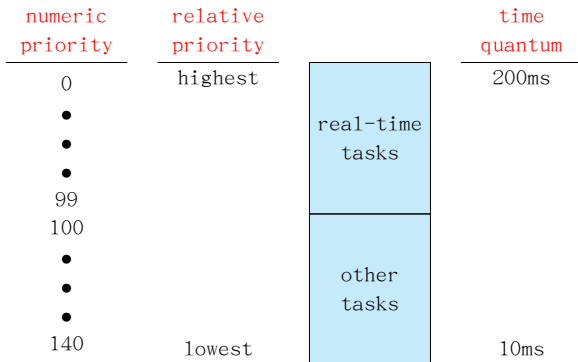
是一组规则，它们决定什么时候以怎样的方式选择一个新进程运行。

Linux 2.6.26中

- SCHED_NORMAL
- SCHED_FIFO (for real-time process)
- SCHED_RR (for real-time process)
- SCHED_BATCH
- SCHED_IDLE

Priorities

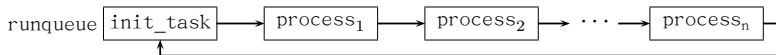
- The **Linux scheduler**: **preemptive, priority-based**
 - two separate priority ranges: lower value \equiv higher priority
 - real-time range: 0~99
 - a nice value rang: 100~140
 - **higher**-priority \Rightarrow **longer** time quanta
(Unlike Solaris and Windows XP)



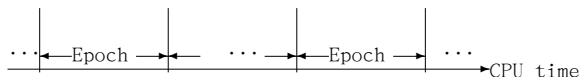
- The **Linux scheduler**: **preemptive, priority-based**
 - two separate priority ranges: lower value \equiv higher priority
 - real-time range: 0~99
 - a nice value rang: 100~140
- **higher**-priority \Rightarrow **longer** time quanta
(Unlike Solaris and Windows XP)
- **Dynamic priorities**:
scheduler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

- Linux 2.4 scheduler

- need to traverse the runqueue, $O(n)$

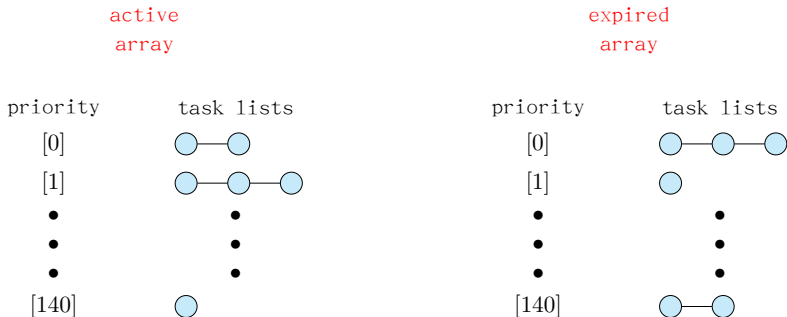


- Epoch**, **default time slice** (基本时间片), **dynamic priorities**



Linux scheduling algorithms

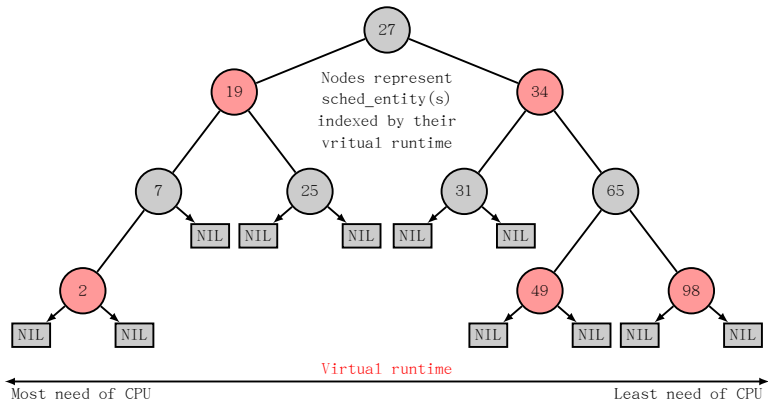
- Linux 2.6.17 scheduler (<2.6.23)
 - O(1)**
 - Double priority-based arrays (双队列): **active** & **expire**



List of tasks indexed according to priorities

Linux scheduling algorithms

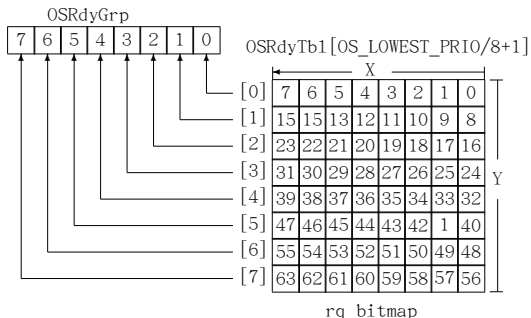
- Linux 2.6.26 scheduler ($\geq 2.6.23$)
 - $O(1)$
 - non-real-time: Complete-Fair-Scheduling (CFS, 完全公平调度), `vruntime`, red-black tree (红黑树)
 - real-time: priority arrays



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

- Priority-based scheduler

- MAX Tasks: 64
- priority number: 0~63



$0 \leq \text{task's priority} < 63$

0	0	Y	Y	Y	X	X	X
---	---	---	---	---	---	---	---

7 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

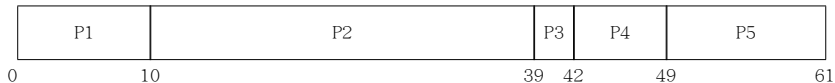
1. Deterministic Modeling(确定性建模) I

- **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)

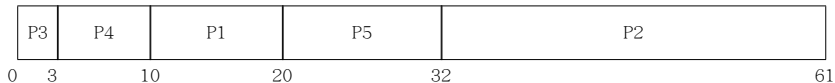
<u>Process</u>	<u>BurstTime</u>
P1	10
P2	29
P3	3
P4	7
P5	12

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 (\equiv waiting queue)
 - Given arriving rates and service rates \Rightarrow 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):

$$n = \lambda \times W$$

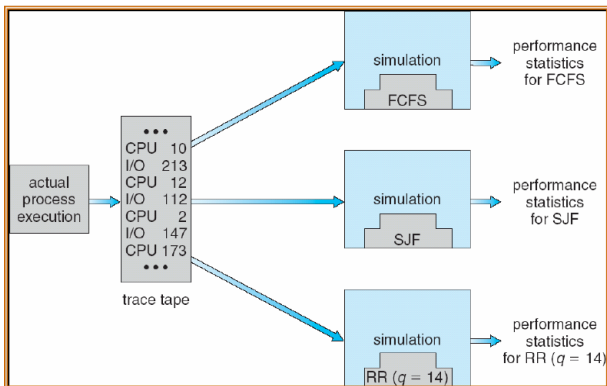
- Little formula is particularly useful because it is valid for any scheduling algorithm and arrival distribution.
- If we know two of the three variables, we can use Little formula to compute the other one.

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 statistics are gathered
- How to generate the data to drive the simulation?
 - **distribution-driven simulation**
 - **random-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)



evaluation 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

8 小结和作业

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 小结和作业

- 参见课程主页

谢谢！