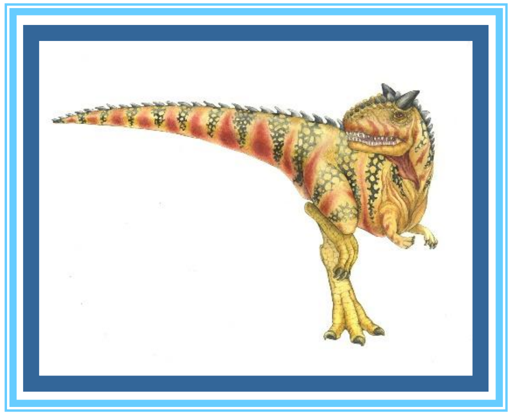
Chapter 5: CPU Scheduling



Chapter 5: CPU Scheduling

上节内容

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms

本节:

- Thread Scheduling
- Multi-Processor Scheduling
- Real-Time CPU Scheduling
- Operating Systems Examples
- Algorithm Evaluation

Operating System Examples

Linux scheduling

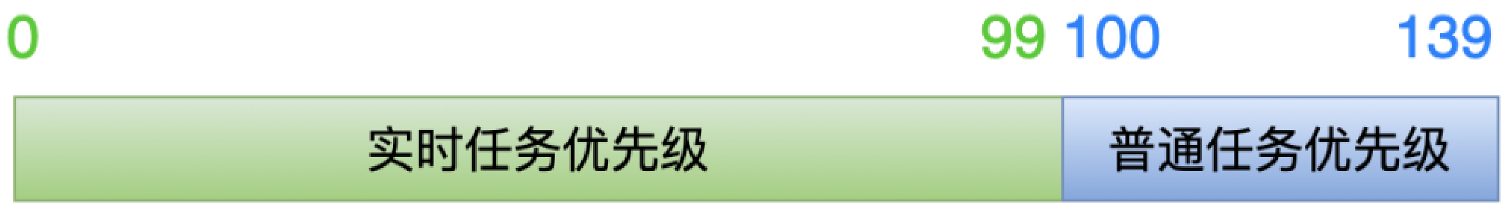
Windows scheduling

Solaris scheduling

Linux Scheduling Through Version 2.5

- Prior to kernel version 2.5, ran variation of standard UNIX scheduling algorithm
- Version 2.5 moved to constant order *O*(1) scheduling time
  - Preemptive, priority based
  - Two priority ranges: time-sharing and real-time
  - **Real-time** range from 0 to 99 and **nice** value from 100 to 140
  - Map into global priority with numerically lower values indicating higher priority
  - Higher priority gets larger q
  - Task run-able as long as time left in time slice (**active**)
  - If no time left (**expired**), not run-able until all other tasks use their slices
  - All run-able tasks tracked in per-CPU **runqueue** data structure
    - Two priority arrays (active, expired)
    - Tasks indexed by priority
    - When no more active, arrays are exchanged
  - Worked well, but poor response times for interactive processes

Linux Scheduling



抢占的、基于优先级的

常数调度时间

两个独立的优先级范围：

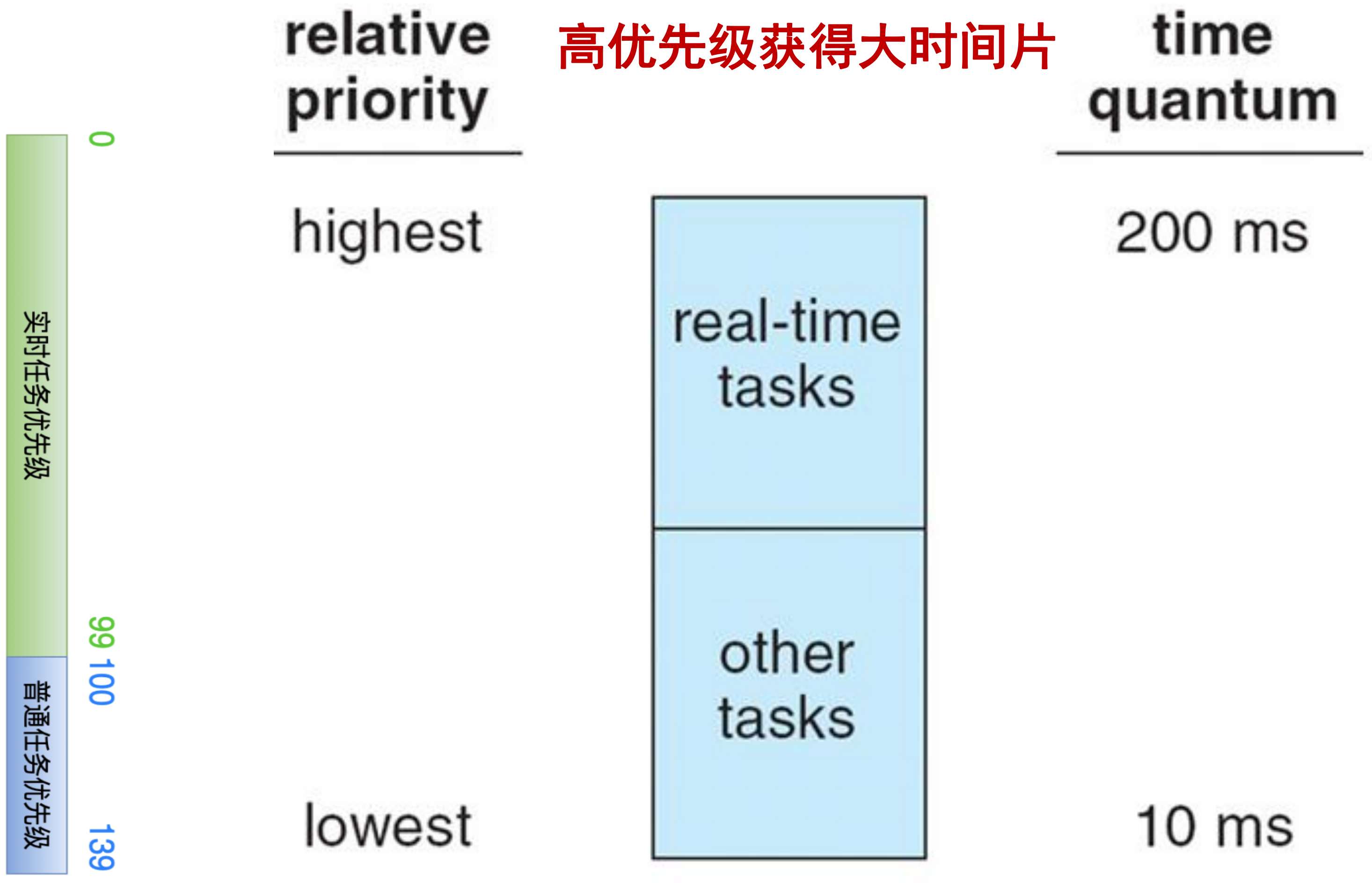
- 实时进程 : FCFS or RR
- 普通进程 : other

Global priority: **数值越小优先级越高**

0~139: 内核角度的优先级

Priorities and Time-slice length

高优先级获得大时间片



在程序中、或使用命令如何设置优先级

1: 实时进程：静态优先级

```
struct sched_param param；
param. _sched_priority ＝ xxx； //创建线程时设置线程优先级
```

当创建线程函数进入内核层面的时候，内核计算真正的优先级数值：

```
kernel priority ＝ 100 -1 - param. _sched_priority
```

2：普通进程 ***（修改 nice 值）***

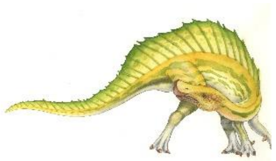
Nice：设定进程相对优先级

Renice：重新设定优先级

-20~19 共40个级别，nice值越大，优先级越低

优先级会被内核动态重新计算

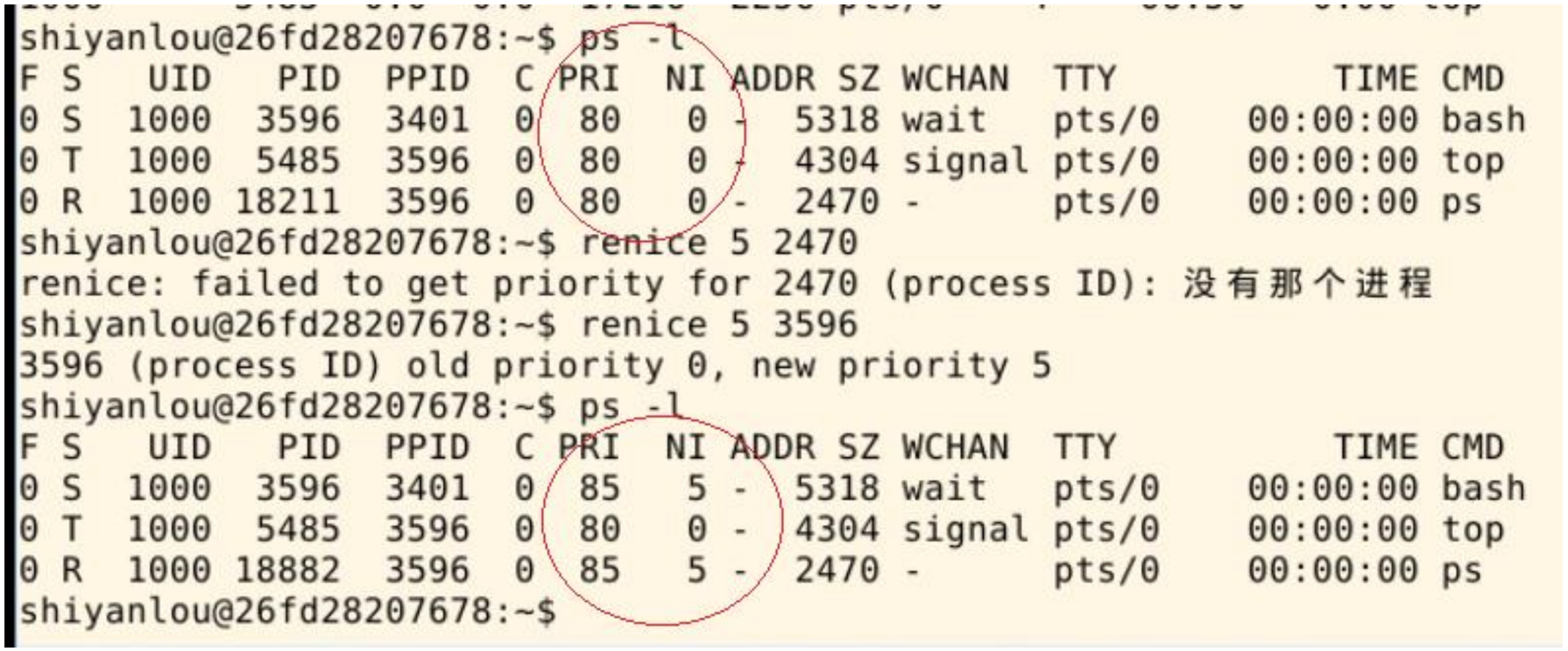
睡眠时间越长的一般交互性越高



Operating System Examples

Linux scheduling

What is PRI and NI ?



```
shiyanlou@26fd28207678:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  3596  3401  0  80   0 -  5318 wait   pts/0    00:00:00 bash
0 T  1000  5485  3596  0  80   0 -  4304 signal pts/0    00:00:00 top
0 R  1000 18211  3596  0  80   0 -  2470 -      pts/0    00:00:00 ps
shiyanlou@26fd28207678:~$ renice 5 2470
renice: failed to get priority for 2470 (process ID): 没有那个进程
shiyanlou@26fd28207678:~$ renice 5 3596
3596 (process ID) old priority 0, new priority 5
shiyanlou@26fd28207678:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  3596  3401  0  85   5 -  5318 wait   pts/0    00:00:00 bash
0 T  1000  5485  3596  0  80   0 -  4304 signal pts/0    00:00:00 top
0 R  1000 18882  3596  0  85   5 -  2470 -      pts/0    00:00:00 ps
shiyanlou@26fd28207678:~$
```

kernel priority = 100 + 20 + nice

nice 的合法数值是： - 20 ~ 19。

Linux Scheduling (2.6以前)

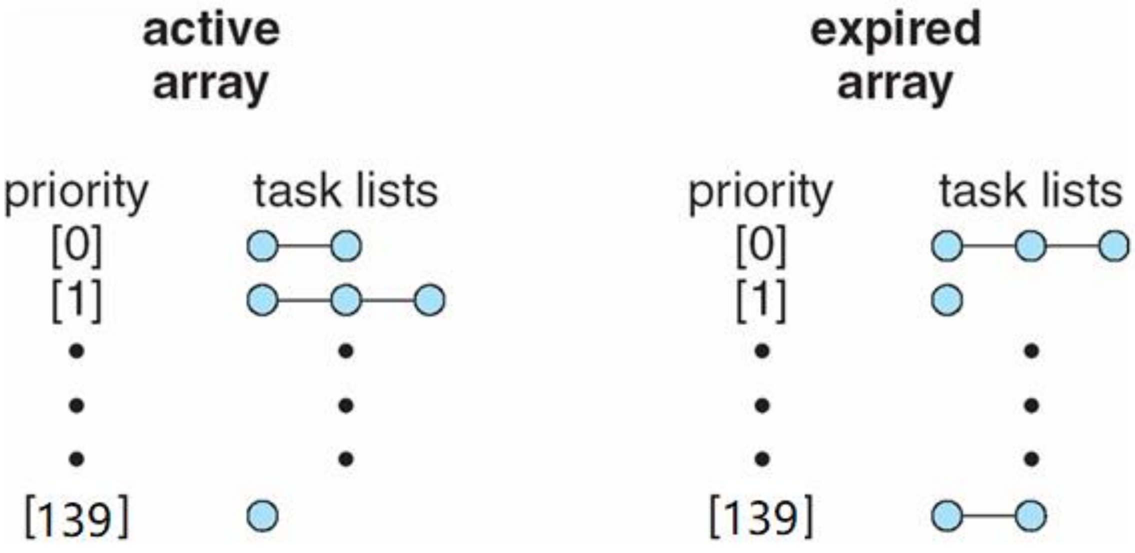
O(1) 调度程序性能好，但交互进程的响应时间欠佳。

可以运行的任务(**active**)

运行 time slice 中剩余的时间片

到期任务 (**expired, 到期,** 时间片运行完),

不能再执行，除非其他的任务也把各自的时间片用完



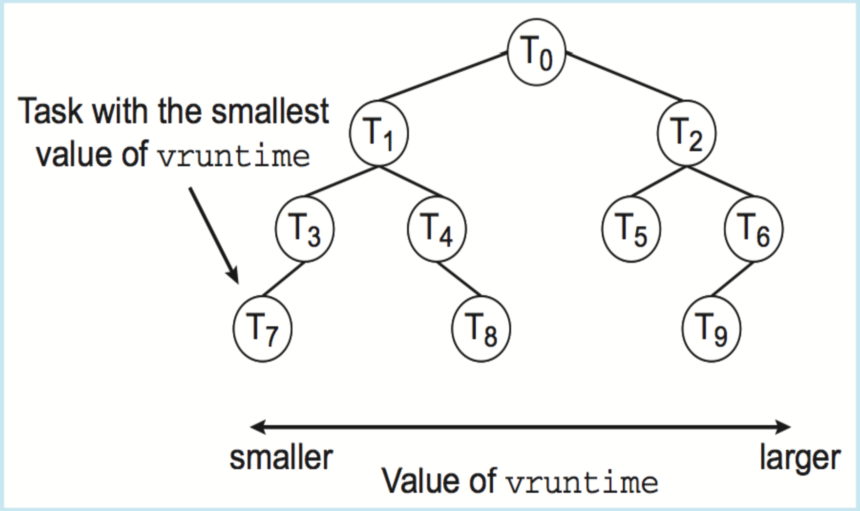
当活动队列空，两个队列交换

Linux Scheduling in Version 2.6.23 +

- ***Completely Fair Scheduler*** (CFS) 完全公平的调度器
- **Scheduling classes**
  - Each has specific priority
  - Scheduler picks highest priority task in highest scheduling class
  - Rather than quantum based on fixed time allotments, based on proportion of CPU time
  - 2 scheduling classes included, others can be added
    1. default
    2. real-time
- Quantum calculated based on **nice value** from -20 to +19
  - Lower value is higher priority
  - Calculates **target latency** – interval of time during which task should run at least once
  - Target latency can increase if say number of active tasks increases
- **CFS scheduler** maintains per task **virtual run time** in variable **`vruntime`**
  - Associated with decay factor based on priority of task – lower priority is higher decay rate
  - Normal default priority yields virtual run time = actual run time
- To decide next task to run, scheduler picks task with lowest virtual run time

CFS Performance

The Linux CFS scheduler provides an efficient algorithm for selecting which task to run next. Each runnable task is placed in a red-black tree—a balanced binary search tree whose key is based on the value of `vruntime`. This tree is shown below:



When a task becomes runnable, it is added to the tree. If a task on the tree is not runnable (for example, if it is blocked while waiting for I/O), it is removed. Generally speaking, tasks that have been given less processing time (smaller values of `vruntime`) are toward the left side of the tree, and tasks that have been given more processing time are on the right side. According to the properties of a binary search tree, the leftmost node has the smallest key value, which for the sake of the CFS scheduler means that it is the task with the highest priority. Because the red-black tree is balanced, navigating it to discover the leftmost node will require $O(lg N)$ operations (where N is the number of nodes in the tree). However, for efficiency reasons, the Linux scheduler caches this value in the variable `rb_leftmost`, and thus determining which task to run next requires only retrieving the cached value.

CFS 基本原理：
设定一个调度周期（*sched_latency_ns*），目标是让每个进程在这个周期内至少有机会运行一次，即每个进程等待CPU的时间最长不超过这个调度周期；
然后根据进程的数量，平分这个调度周期内的CPU使用权，由于进程的优先级即nice值不同，分割调度周期的时候要加权；
每个进程的累计运行时间保存在自己的vruntime（进程的虚拟运行时间）字段里，哪个进程的vruntime最小就获得本轮运行的权利。

参考阅读

Linux进程调度技术的前世今生_confirmwz的博客-CSDN博客

多项选择题

Regarding to the Linux Scheduling, which is (are) correct? (Multiple Selections) 下面关于Linux调度的说法，哪些是正确的?

A. Real time processes have the highest priorities

实时的进程优先级最高

B. Higher priority processes have shorter time quantum

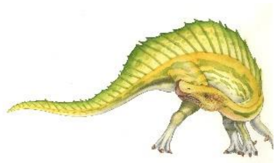
优先级越高的进程时间片越短

C. Starvation can be avoided by adjusting the nice values

可以通过调整nice值来避免出现饥饿

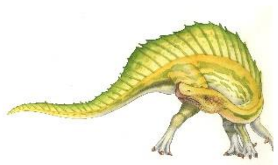
D. The priority of a normal process can be 99

一个普通进程的优先级可以为99



Windows Scheduling

- Windows uses priority-based preemptive scheduling
- Highest-priority thread runs next
- **Dispatcher** is scheduler
- Thread runs until (1) blocks, (2) uses time slice, (3) preempted by higher-priority thread
- Real-time threads can preempt non-real-time
- 32-level priority scheme
- **Variable class** is 1-15, **real-time class** is 16-31
- Priority 0 is memory-management thread
- Queue for each priority
- If no run-able thread, runs **idle thread**



Windows Priority Classes

Win32 API identifies several priority classes to which a process can belong

- REALTIME_PRIORITY_CLASS, HIGH_PRIORITY_CLASS, ABOVE_NORMAL_PRIORITY_CLASS,NORMAL_PRIORITY_CLASS, BELOW_NORMAL_PRIORITY_CLASS, IDLE_PRIORITY_CLASS
- All are variable except REALTIME

A thread within a given priority class has a relative priority

- TIME_CRITICAL, HIGHEST, ABOVE_NORMAL, NORMAL, BELOW_NORMAL, LOWEST, IDLE

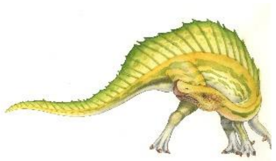
Priority class and relative priority combine to give numeric priority

Base priority is NORMAL within the class

If quantum expires, priority lowered, but never below base

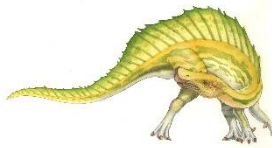
Windows Priority Classes (Cont.)

- If wait occurs, priority boosted depending on what was waited for
- Foreground window given 3x priority boost
- Windows 7 added **user-mode scheduling** (**UMS**)
  - Applications create and manage threads independent of kernel
  - For large number of threads, much more efficient
  - UMS schedulers come from programming language libraries like C++ **Concurrent Runtime** (ConcRT) framework



Windows Priorities

	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



End of Chapter 5