
# OPERATING SYSTEMS

## Case Study: Linux/ Windows Scheduling Policies.

**Suresh Jamadagni**
Department of Computer Science

# OPERATING SYSTEMS

## Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

# OPERATING SYSTEMS

## Case Study: Linux/ Windows Scheduling Policies

**Suresh Jamadagni**

Department of Computer Science

# OPERATING SYSTEMS

## Linux Scheduling Through Version 2.5


- Process Scheduling in Linux
- Linux kernel ran a variation of standard UNIX scheduling algorithm
- It did not support for SMP systems
- It had poor performance for larger processes

## Linux Scheduling Version 2.5


- Kernel moved to constant order *O*(1) scheduling time
- Supported SMP systems - processor affinity and load balancing between processors with good performance
- Poor response times for the interactive processes that are common on many desktop computer systems

# OPERATING SYSTEMS

## Linux Scheduling in Version 2.6.23 +


- Completely Fair Scheduler (CFS) is the default scheduling algorithm.
- Based on **Scheduling classes**
  - Each class is assigned a specific priority.
  - Using different scheduling classes, the kernel can accommodate different scheduling algorithms
  - Scheduler picks the highest priority task in the highest scheduling class
  - Two scheduling classes are included, others can be added
    1. A scheduling class with CFS algorithm
    2. A real-time scheduling class

# Linux Scheduling in Version 2.6.23 + (Cont.)


**Completely Fair Scheduler (CFS)**

- CFS scheduler assigns a **proportion of CPU processing time** to each task.
- Proportion calculated based on **nice value** which ranges from -20 to +19
  - A numerically lower nice value indicates a higher relative priority.
  - Tasks with lower nice values receive a higher proportion of CPU processing time than tasks with higher nice values
- Calculates **target latency** – interval of time during which task should run at least once
  - Proportions of CPU time are allocated from the value of targeted latency computed.
  - Target latency can increase if number of active tasks increase

# Linux Scheduling in Version 2.6.23 + (Cont.)

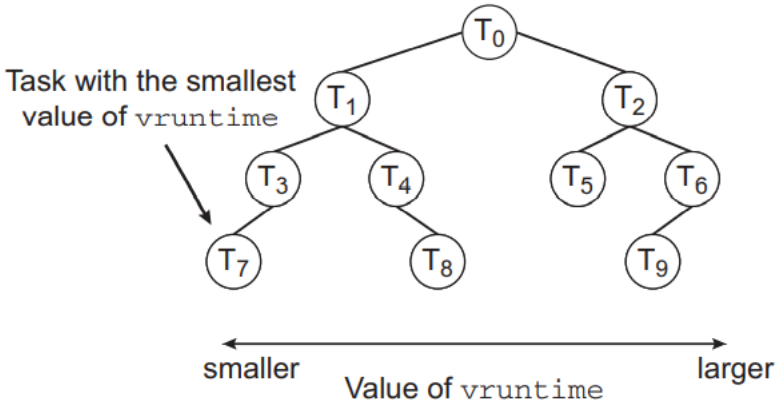

## Completely Fair Scheduler (CFS)

- CFS scheduler doesn't directly assign priorities
- 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

## Linux Scheduling in Version 2.6.23 + (Cont.)

### Completely Fair Scheduler (CFS)

- Each runnable task is placed in a balanced binary search tree whose key is based on the value of **vruntime**
- When a task becomes runnable, it is added to the tree
- When a task is not runnable, it is deleted from the tree
- Navigating the tree to discover the task to run (leftmost node) will require O(lg N) operations (where N is the number of nodes in the tree).



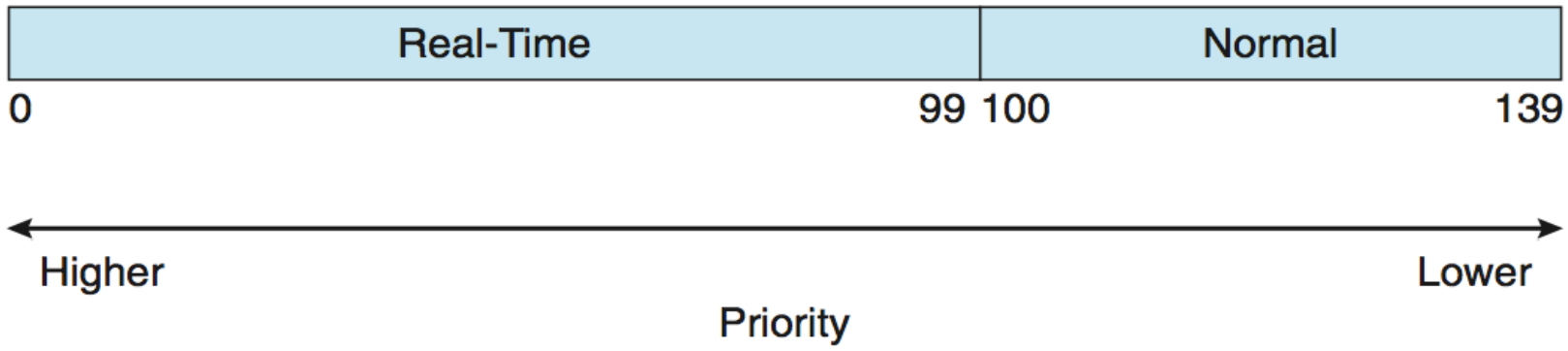
## Linux Scheduling in Version 2.6.23 + (Cont.)


### Completely Fair Scheduler (CFS)

- Assume that two tasks have the same nice values.
- One task is I/O-bound and the other is CPU-bound
- The value of **vruntime** will be lower for the I/O-bound task than for the CPU-bound task, giving the I/O-bound task higher priority than the CPU-bound task.
- If the CPU-bound task is executing when the I/O-bound task becomes eligible to run, the I/O-bound task will preempt the CPU-bound task.

## Linux Real Time Scheduling

- Real-time scheduling according to POSIX.1b
  - Real-time tasks have static priorities (0-99)
- Real-time plus normal map into global priority scheme
- Nice value of -20 maps to global priority 100
- Nice value of +19 maps to priority 139



# OPERATING SYSTEMS

## Windows Scheduling


- Uses priority based pre-emptive scheduling algorithm
- Scheduler ensures that the **highest-priority thread** will always run
- Windows kernel that handles scheduling is called the **dispatcher**
- Thread selected by the dispatcher runs until it is preempted by a higher-priority thread or until it terminates or until its time quantum ends or until it calls a blocking system call
- Dispatcher uses a 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 class
- If no run-able thread, runs **idle thread**

## Windows Priority Classes


- Windows 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

# OPERATING SYSTEMS

## Windows Thread Priorities

- A thread within a given priority class also has a relative priority
- Priority of each thread is based on both the priority class it belongs to (top row in the diagram) and its relative priority within that class (left column in the diagram)

| | 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 |

## Windows Priority Classes (Cont.)

- If wait occurs, priority boosted depending on what was waited for
- Windows distinguishes between foreground process and background processes
- Foreground processes given 3x priority boost
- This priority boost gives the foreground process three times longer to run before a time-sharing preemption occurs.


# THANK YOU

**Suresh Jamadagni**
Department of Computer Science Engineering

**sureshjamadagni@pes.edu**