EECS 750: Advanced Operating Systems

01/27/2014

Heechul Yun
Administrative

• Next summary assignment due
  – Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin, PPoPP '09
  – by 11:59 p.m., Tuesday

• Sign up for presentations
  – First student presentation on Feb 3.
Today

• Topic: Unicore CPU scheduling

• Linux CPU schedulers
  – A brief history

• Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler [SOSP’99]
CPU Scheduling

• OS abstraction: many processes
• H/W reality: one processor (assume uncore for now)

• What is CPU scheduling?
  – Deciding which task to run, on what time, and how long.
  – Many possible ways ➔ many CPU schedulers

• Undergraduate OS question
  – What’s the difference between preemptive vs. non-preemptive scheduling?
CPU Scheduling Goals

• Fairness
  – “Everyone should get an equal chance to succeed.”

• Responsiveness
  – “If I press the ok button, the screen should be updated immediately”

• Throughput
  – “as much job done as possible.”
Linux Scheduling Framework

- Completely Fair Scheduler (CFS)
- Real-time fixed priority scheduler
Linux Scheduler: A Brief History

- v2.2 and before
  - Simple round robin policy
    - Not fair, why?
- v2.4
  - O(N) scheduler
    - Fair (mostly), doesn’t scale well
- v2.6~v2.6.22
  - O(1) scheduler
    - Scale well, but rather complex and use heuristics to handle task interactivity, and not fair
- V2.6.23~
  - CFS scheduler (Completely Fair Scheduler), 2007
    - O(logN), simple, solid theoretical support, and fair as the name suggested
Linux 2.4: O(N) Scheduler

• Divide time into epochs. In each epoch, every process gets a specified time quantum to spend on that epoch

• At every scheduling event, it computes “goodness” of each task, pick the task with highest “goodness” value by iterating the entire task list
  – Goodness is based on the remaining time quantum + priority
  – Scheduling events: I/O, scheduler tick timer interrupt, yield

• It doesn’t scale. Why?
Linux 2.6: O(1) Scheduler

• A kind of multi-level priority queueing system
  – Each priority has queues

• Pick the first runnable task with the highest priority
  – Therefore, O(1)

• Use **heuristics** to dynamically boost the priority of interactive tasks
  – consider the average sleep time of each task
Completely Fair Scheduler (CFS)

• Each task maintains its virtual time
  \[ V_i = E_i \times \frac{1}{w_i}, \] where \( E \) is executed time, \( w \) is a weight

• Pick the task with the **smallest virtual time**
  – Tasks are sorted according to their virtual times
  – Managed by a **red-black tree**, \( O(\log N) \)
Completely Fair Scheduler (CFS)

- **Red-black tree**
  - Self-balancing binary search tree
  - Insert: $O(\log N)$, Remove: $O(1)$

Figure source: M. Tim Jones, “Inside the Linux 2.6 Completely Fair Scheduler”, IBM developerWorks
Completely Fair Scheduler (CFS)

• Guarantee **fairness** and **bounded latency**
  – No complex heuristics for interactivity. Why?
Today’s Paper

• “Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose”, SOSP’1999
  – Side note: it was even before the O(N) scheduler of Linux 2.4.0 (2001)
The Problem

• (Interactive) real-time applications
  – VoIP, video and audio decoding
  – GUI applications
  – Google search query processing
    • Thousands of machines are processing in parallel with different indexes
    • Results that exceeds a certain deadline (2~300ms) will be discarded

• How to satisfy their real-time requirements?
Other Solutions

• Priority schedulers
  – Run high priority task first no matter what
  – Problems?

• Earliest Deadline First (EDF) scheduler
  – Pick a task with the earliest deadline
  – Each task’s **period** $P$ and **computation time** $C$ should be known in advance
  – Can provide **temporal isolation**
  – Linux 3.14 will integrate a deadline scheduler for the first time
  – Popular in the academic (real-time) community, but not so much in industry (so far). Why?
Borrowed Virtual Time (BVT) Scheduler

• In short
  – **Weighted Fair Sharing** + alpha
  – Alpha = borrowing virtual time from future, up to a certain limit
    • To reduce latencies of real-time applications
  – Requires users to specify how much **virtual time to borrow** and the **limits**
  – That’s it

• Author’s claim
  – BVT can handle both normal and real-time tasks
Some Background

• Generalized Processor Sharing (GPS) [TON’93]
  – Idealized fluid model for sharing bandwidth (network) among multiple flows
  – Bandwidth is divided fairly according to flow weights
• Weighted Fair Queuing
  – Packetized approximation of GPS
• GPS ≈ WFQ ≈ CFS ≈ Weighted Fair Sharing in BVT
Weighed Fair Sharing: Example

Weights: gcc = 2/3, bigsim=1/3

X-axis: mcu (tick), Y-axis: virtual time

Fair over a scheduling window of 9 mcu
Weighted Fair Sharing

- How to set the virtual time of a new task?
  - Can’t set as zero. Why?
  - System virtual time (SVT)
    - The minimum virtual time among all active tasks
  - The new task can “catch-up” tasks by setting its virtual time with SVT
Weighed Fair Sharing: Example 2

Weights: gcc = 2/3, bigsim=1/3
X-axis: mcu (tick), Y-axis: virtual time
gcc slept 15 mcu
Borrowing Virtual Time

• Borrowing virtual time from the future
  – \( E = A - \text{warpBack} \)
    • \( E \): effective virtual time
    • \( A \): actual virtual time
    • \( \text{warpBack} \): borrowed virtual time
  – Reduce effective virtual time \( \rightarrow \) scheduled earlier
  – For low-latency dispatch of real-time apps.
BVT: Example

The diagram shows the results of a benchmark with three different tools: gcc, bigsim, and mpeg. The x-axis represents the warpBack parameter, and the y-axis represents the performance metric. The mpeg tool is indicated to have a setting of mpeg=50.
BVT

- Preventing “reckless borrowing”
  - L: warp time limit (in real-time)
    - to prevent starvation
  - U: unwarp time requirement (in real-time)
    - to prevent periodic task exceeding its fair share
Evaluation Results

Generated real-time workloads (randres) + background (cont)
Round Robin = WRR = BVT(warp=0) = CFS
Summary

• Understand:
  – Linux’s Completely Fair Scheduler (CFS)
  – Borrowed Virtual Time scheduler in SOSP’99 paper
Discussion

• Similarities and differences between CFS and BVT

• Dual-schedulers (Linux) vs. single all-round scheduler (BVT)
  – Linux: Real-Time scheduler + CFS
  – BVT: one scheduler handle both