## Scheduling Support for Multimedia Applications

- "A Hierarchical CPU Scheduler for Multimedia Operating Systems"
  OSDI'96, Boyal, Guo, and Vin
- "The Design, Implementation, and Evaluation of SMART: A Scheduler for Multimedia Applications"
  SOSP'97, Nieh and Lam
- "Scheduling Policies to Support Multimedia Applications"
  SIGMETRICS'98, Nguyen and Zahorjan



## In General...

- Focus: desktop
- HCI via images, video, audio, interactive graphics
- Hardware support
- Software support hasn't kept pace
  - problems
  - video jitter
  - lip-synch of audio and video
  - slow interactive response time



## Types of demands

- real time, interactive, and batch applications
- hard-real time applications (earliest deadline first, rate monotonic scheduling)
- soft-real time applications (statistically QoS)
- "insatiable" resource demands and overload (e.g., playback window)
- dynamically adaptive applications (e.g., skip frames or display at lower image quality)
- conventional applications
- native OS demands
- dynamic environment (load)
- user preferences (think of importance)



## Hierarchical CPU Scheduler

- Foci: three application classes
- Hard-real time applications
- Soft-real time applications
- Best effort applications
- Qualities of good scheduler
- *Hierarchical partitioning of CPU bandwidth*
  - fair bandwidth distribution
  - no a priori knowledge
  - TPUT guarantees
  - efficient implementation
- Tree structure

- Slices of processor bandwidth as *weights*
- Different scheduling policies per node (depending on policy)
- Partition of bandwidth allocated to a class such that each class gets a share
- Allocated residual bandwidth fairly among its subclasses



## Start-time Fair Queuing

- Target: fair cpu allocation
- For every thread, the normalized work (by weight) is the same
- Start tag
- Increasing order of start tags
- Definitions
  - $q_f^j$: $j^{th}$ quantum of thread $f$
  - $l_f^j$: length of $j^{th}$ quantum of thread $f$
  - $A(q_f^j)$: time the $j^{th}$ quantum is requested
  - $v(t)$: virtual time
    the clock does not go forward while threads are blocked



## Algorithm

1. Start tag $S_f$ = max( $v(A(q_f^j)), F_f$)
   $F_f$ is initially 0...
   when the $j^{th}$ quantum finishes it is incremented as:
   $F_f = S_f + l_f^j / r_f$
   where $r_f$ is the weight of the thread $f$.
2. If CPU is busy, then $v(t)$ is equal to the start tag of thread at time $t$.
   If CPU is idle, $v(t)$ is equal to the max finish tag of any thread.
3. Threads are serviced in increasing start tag order, ties are broken arbitrarily.



## Properties...

- Fair allocation regardless if variation in available processing bandwidth
- Does not require the length of the quantum to be known a priori
- Bounds on maximum delay and minimum TPUT achieved
  (proof analogous to network bandwidth)
- Computationally efficient
- Implementation: actual scheduling occurs recursively
- A node is scheduled iff it has one runnable thread
- Finish and start tags of ancestor nodes must be updated
- SFQ can be used as either node of leaf scheduler

## SMART: a Scheduler for Multimedia Applications

- Supports: real-time, interactive, and conventional computations
- Basic idea: sheds real time tasks and regulates execution rates when overloaded
- Good in underload
- At overload, does not lock-up the system (like UNIX SVR4)



## Basic Ideas...

- Two decisions
  - importance (overall resource allocation)
  - urgency (when each task i s to be given its allocation)
- Importance: real time and conventional tasks are based on priorities and weighted fair queuing
- Urgency: earliest deadline first
- Bias on conventional tasks towards more varied service latency (to better serve interactive and real time during transient loads)



## Overview of Interface and Usage

- Allows applications to communicate their timing requirements with the system
- Real time computations
  - time constraints: (dealing, estimate of required processing time to meet the deadline)
  - notifications: informs the scheduler that deadline cannot be met
- User can define
  - different priorities per applications
  - different “shares” per applications (equivalent to “weights” of the hierarchical scheduler)



## Smart Objectives

- Priority: never degrade performance on a high priority application in the presence of a low priority
- Sharing: proportionally for real time and conventional tasks (if in the same priority class)
- Transient load: graceful transitions, adapting to load of apps that come and go...
- Underload: satisfy real time constraints and fast interactive R.T.
- Unfairness: tolerate instantaneous unfairness
- Resource availability: notification

## Urgency *vs* Importance

- Importance: (priority, biased virtual finishing time BVFT)
- $task_A > task_B$ if priority $A$ ¿ priority $B$
- $task_A > task_B$ if $A$ and $B$ have some priority, but $BVFT_A$ ¿ $BVFT_B$
- If the task with the highest value is a conventional one, schedule it
- Else
  - candidate set of real time tasks with higher value
  - best effort real time scheduling
  - if it cannot complete before deadline, send a notification



- Virtual time for task:
  $v(t) = v(t) + \frac{t-r}{S}$, where $r$ is the initiated timestamp
- Queue virtual time:
  $V_p(t) = V_p(r) + \frac{t-r}{\sum S_a}$
- Organize queues by properties:

  1. $v(t) = V_p(t)$ then task received its proportion
  2. $v(t) < V_p(t)$ then task received less than its share
  3. $v(t) > V_p(t)$ then task received more than its share

- $\beta(t)$: biased virtual finishing time of task $t$
- $\beta(t_0) = V_p(t_0) + \frac{Q}{S}$, where $V_p(t_0)$ is the queue virtual time and $Q$ is the time quantum



- add a bias when the task executes for a time quantum:
  $\beta(t) = \beta(t_1) + \frac{Q}{S} + \frac{b}{S}$
- basically, delay long running tasks to serve suddenly bursty workloads
- blocking for I/O or other events: allows a blocked task to remain in its priority queue for a limited time
- the task may reenter the queue and its BVFT is adjusted