Introduction

Supporting Time-Sensitive Applications on a Commodity OS

by: Ashvin Goel, Luca Abeni, Charles Krasic, Jim Snow, Jonathan Walpole

presented by: Dora Raymaker
Background: Real-Time and Time Sensitive Applications

- **Real-Time Systems**
  - Must meet an exact deadline
  - Failure to meet deadline means system failure

- **Time Sensitive Applications**
  - Application with real-time requirements
  - Examples: A/V media, soft modems
  - Emphasis on:
    - Timing – time sensitive applications
    - Volume – throughput applications
Background: Relationship to OS

- **Time-Sensitive Applications Need**
  - Low latency
  - Precise timing
  - Effective scheduling

- **Especially Affects Operating System in**
  - Timers and interrupts
  - Preemption
  - CPU scheduling
  - Resource sharing
Motivation

- Commodity systems not good at time-sensitive applications
- Real-time systems not good at throughput applications
- OS increasingly need to handle both time sensitive and throughput applications
- Attempts to compromise not holistic enough in solution approach
Goal

- Time Sensitive Linux (TSL)

- Create a version of Linux that can:
  - Effectively run time-sensitive applications
  - Effectively run throughput applications
  - Be implemented without modifying existing applications

- Focus on reducing latency
Problem: Kernel Latency

- Reduction of Over-All Kernel Latency
  - Three areas of latency
  - Three solutions needed
Solution: Three Hybrid Designs
Timer Latency: One-Shot Timers

- Problem with Periodic Timers
  - too often -> lots of interrupt overhead
  - too sparse -> can't do real time

- Solution
  - One-Shot Timer only goes off when needed

- Implementation
  - Hardware solution
  - Can have high cost in interrupts
Timer Latency: Soft Timers

- Problem with One-Shot Timers
  - High cost of interrupts

- Solution
  - Soft timers handle events without interrupts
  - Coordinate with CPU

- Implementation
  - Software solution
  - Can have problematic latency
Timer Latency Solution: Firm Timers

- Overshoot
  - high = soft
  - low = one-shot
  - provides upper bound on time before event handled

- Unnecessary interrupts avoided

- Soft timer latency controlled
Preemption Latency: Explicit Preemption

- **Problem**
  - Threads executing in kernel cannot be preempted

- **Solution**
  - Specify yield points in kernel code

- **Pro**
  - No locks

- **Con**
  - Latency maximum time between yields
  - Hard to implement
  - Dependent on system call paths
Preemption Latency: Preemptible Kernel

- **Problem**
  - Threads executing in kernel cannot be preempted

- **Solution**
  - Protect kernel data with locks

- **Pro**
  - Easy to implement
  - Not dependent on system call paths

- **Con**
  - Not efficient if data sections are large
  - Latency maximum time lock held
Preemption Latency Solution: Fine-Grained Kernel Preemptibility

- Robert Love's Linux patch
- Use Explicit Preemption to break up long data sections
- Get a Preemptible Kernel with fine-grained preemptibility
Scheduling Latency: Priority Scheduling (HLP)

- Algorithm (Highest Locking Priority)
  - Highest priority task goes first
  - Task acquires resource
  - Task gets highest priority of any task that can get that resource

- Pro
  - Simple
  - Avoids priority inversion

- Con
  - Does not provide temporal protection
Scheduling Latency: Proportion-Period Scheduling

- **Algorithm**
  - Allocate task fixed proportion $P$ of total CPU time for a given period $T$
  - Allocated time $Q = P \times T$
  - Task executes for $Q$, then blocks or reschedules until the next $P$

- **Pro**
  - Provides temporal protection

- **Con**
  - $P$ and $T$ must be specified
Scheduling Latency Solution: TSL Scheduling

- Both Priority and Proportion-Period used
- Proportion-Period takes precedence
Experimentation: Goals

- Behavior of Time-Sensitive Apps on TSL
  - Mplayer
  - Proportion-Period Scheduler

- Overhead of TSL
  - Cost of fine-grained preemption
  - Cost of executing firm timers

- Interested in realistic scenarios
Experimentation: Mplayer Design

- A/V player
- Synchronization of A/V reveals timing issues
- Compared vs. Linux
- Three Load Tests:
  - non-kernel CPU stress test
  - kernel CPU buffer
  - file CPU file system copy
Experimentation: Mplayer Results

- Figures for kernel CPU buffer test
- Similar results for other Mplayer experiments
- Note scale on Y axis
- Experiment showed necessity of HLP scheduling
Experimentation: Scheduler Design

- Proportion-Period scheduler used by TSL
- Two processes invoke and store time, difference between actual and desired times reveal timing issues
- Measure = actual time – desired time
- Two Load Tests
  - no file system load
  - file system copy
Experimentation: Scheduler Results

- Note figures are max, not avg
- Greater deviation for greater file system load
- Future work will try to improve performance under heavy file system load

<table>
<thead>
<tr>
<th></th>
<th>No Load</th>
<th>File System Load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max Proportion Deviation</td>
<td>Max Period Deviation</td>
</tr>
<tr>
<td>Thread 1</td>
<td>0.3% (≈25 μs)</td>
<td>5 μs</td>
</tr>
<tr>
<td>Proportion: 40%, 3276.8 μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Period: 8192 μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 2</td>
<td>0.7% (≈3 μs)</td>
<td>10 μs</td>
</tr>
<tr>
<td>Proportion: 20%, 102.4 μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Period: 512 μs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experimentation: Preemption Check Overhead Design

- Isolates cost of preemption checking
- Benchmarks stress preemption latency
- Measure = test completion time TSL / test completion time Linux
- Three Tests
  - memory access
  - fork
  - file system access
Experimentation: Preemption Check Overhead Results

- Memory Access
  - TSL overhead .42 +/- .18%

- Fork
  - TSL overhead .53 +/- .06%

- File System Access
  - TSL no significant overhead

- Overhead of checking for preemption points in TSL low
Experimentation: Firm Timer Overhead Design

- Ray tracer (memory, CPU intensive) & get current time (TS) run in 10 ms periods
- Overhead = time to render in TSL / time to render in Linux
- Compared overhead
  - Linux
  - TSL hard timers
  - TSL soft timers
  - TSL firm timers at various offsets
- Firm timers for short period timer
Experimentation: Firm Timer Overhead Results

- Left: Overhead Comparison
  - Similar results for more timers
- Right: Short Period Timers
- Effectiveness of Timer Type Depends
  - Strengths / weaknesses of each type of timer
  - Needs of system
Summary and Future Direction

- TSL is effective for running both time-sensitive and throughput applications
- Latency can be reduced significantly by targeting all areas where it occurs
- Results are promising, but TSL is still experimental
  - Firm timers and timing
  - Network effects
  - Real workloads
Conclusions and Relations: Theory and Abstractions

- Processes
- Events
- Interrupts and timing
- Scheduling
- Concurrency
- Levels of implementation (e.g. HW vs. SW)
- ...
Conclusions and Relations: Practical Considerations

- Idea reuse
- Using weaknesses as strengths
- Hybridization
- Whole systems considerations
- ...

...
Conclusions and Relations: Scientific Process

- Problem identification
- Hypothesis formation
- Requirements and design specification
- Solution implementation
- Experimentation
- Conclusion leads to:
  - Problem identification
  - Hypothesis formation
  - ...

Conclusions and Relations: Higher Order Concepts

- Specialization / generalization
- Trade-Offs
  - Speed vs. accuracy
  - Performance vs. ease
  - Resources
  - ...
- Compromise
- “It depends.”
Thank You!


– Robert Love. The Linux kernel preemption project.  
  http://kpreempt.sf.net.

– Mohit Arong. Soft timers.  

– Beal, David. Linux as a real time operating system.  

– Wikipedia for overview of real time systems