A Flexible Scheduling Framework (for Linux):
Supporting Multiple Programming Models
with Arbitrary Semantics

Noah Watkins, Jared Straub*, Douglas Niehaus*

* University of Kansas

Presented by Noah Watkins
Systems Research Lab
UC Santa Cruz

Adapted from RTLWS09 slides
Overview

• Growing trend toward single systems with wide range of semantics
• Linux is used in many application areas, and is attractive for new research and development
• Priority-based systems have a difficult time supporting multiple, competing semantics
  – Performance management
• Non-priority based scheduling requires general treatment of system components
• *Proxy Execution: General treatment of CC*
Single System, Multiple Semantics
...But why use Linux?

• Economic pressure to select cheap solutions
  – Need strong justification for custom systems
  – Hence increasing popularity of Linux as a standard platform.

• Cost and complexity justify multiple applications sharing HW platforms
  – Multi-core and MHz increases make sharing attractive

• *With multiple applications, satisfying all their constraints becomes complex*
Application Semantic Explosion

- Responsiveness
- TV, Video, Games, Productivity
- Latency, Coordination
- Denial of Service, Distributed Coordination
- QoS
- Experimental Scheduling

SRL Research Symposium, 22 October 2009
Performance Management

- Computations use resources, and this affects their behavior
- Managing performance requires managing many system components
  - CPU (thread scheduling), Disk scheduler
  - Software-based resources (e.g. Buffer Cache)
- One application has no competition
  - Ignoring system-level computations
Performance Management

- Real systems have multiple applications, with a range of semantics
- Computations compete with each other for shared resources
  - CPU, Disk, Network
  - SW-based (e.g. buffer cache)
- Managing the performance of the system requires that the interaction among computations be managed

Diagram:
- US
  - App
  - Daemon
  - SSH
  - Monitoring
- OS
  - CPU (scheduling)
  - Disk
  - SWR1
  - Buffer Cache
  - SWR2
Performance Management

- Multiple applications with multiple semantics share many resources
- Servers multiplex client connections with competing policies (e.g. QoS)
- Context-borrowing computations under hard-wired scheduling policies
- *Managing interaction among computations requires managing semantic/policy conflicts*
Goal: Precise Computation Control – It’s Easy, Right?

High Priority

Low Priority
Semantic Mappings: A Developers Job

Set of Semantics

Arbitrary Semantics

F()

Semantic Mapping

Static Priority

Static Priority Scheduling
Semantic Mappings

- Application developers map their semantics onto priority-based PM.
- Complex mappings are difficult to create, understand, model, and verify.
- Developers have no other choice.
  - Priority is ubiquitous and well-understood.
  - Application developers lack knowledge and resources to create new thread scheduler.

Diagram:
- Static-priority PM [+MW]
- Applications: App, Daemon
- SSH
- App Monitoring
- OS
  - Hard-IRQ, Soft-IRQ, Tasklet, etc...
    (in PREEMPT_RT)
  - Disk
  - SWR1
  - Buffer Cache
  - SWR2
  - CPU (scheduling)
Semantic Mapping: Problems Masked

**Logical representation of system semantics**

- **S1**
  - **S2**
    - App1, IRQ 1,2
  - **S3**
    - App 2, 3, 4, IRQ 2,7
  - **S4**
    - Everything else...

**Reality: complex mappings, priority overlaps**

- **High Prio** to **Low Prio**
  - Hard/Soft-IRQs, Tasklets, etc...
  - Application 1
  - App 2, 3, 4
  - Everything else...

SRL Research Symposium, 21 October 2009
Semantic Integration

- So how do we manage shared resources with many concurrently existing semantics?
- A resource is generally built in support of an assumed system semantics
  - E.g. priority-aware implementations
- Semaphores commonly manage access to shared resource
  - Integrated with scheduling via PI protocol

SRL Research Symposium, 21 October 2009
Solution: Directly Represent Scheduling Semantics

- **Group Scheduling**
  - A particular solution
  - Hierarchic scheduling framework at KU
- Represent semantics *directly*
  - No mappings, application scheduling state directly fuels schedulers
- Relationship between application semantics explicitly represented by the hierarchy structure
Direct Representation: Frame Progress

- Multiple pipelines processing frames
- Each pipeline has different performance characteristics
- Goal: Pipelines finish processing frames in sync

- Can be done with user-space concurrency control
- Locks are used for their scheduling affects

- Instead, directly represent the pipeline progress (application state) within the scheduler
- Clear, unambiguous, easily modeled implementation
Integration Difficulty

Directly representing semantics requires general integration of system components

App1, IRQ 1,2

App 2, 3, 4, IRQ 2, 7

Everything else...

US

App

Daemon

SSH

App

Monitoring

OS

Hard-IRQ, Soft-IRQ, Tasklet, etc...

(in PREEMPT_RT)

HW Resource

Concurrency Control

Thread Scheduler

SWR1

Buffer Cache

SWR2
Concurrency Control Integration

Directly representing semantics requires general integration of system components

App1  IRQ 1,2  S1  S2
App 2,3,4  IRQ 2,7  S3
S4
Everything else...

US
App
Daemon
SSH
App
Monitoring

OS
Hard-IRQ, Soft-IRQ, Tasklet, etc...  HW Resource
Thread Scheduler  SWR1  Buffer Cache  SWR2
(in PREEMPT_RT)  Concurrency Control

SRL Research Symposium, 21 October 2009
Concurrency Control Integration

• Common approaches assume scheduling semantics
  – Priority inheritance
  – BWI
    – A semaphore hard-codes this assumption into its implementation
• Directly represented scheduling semantics may use arbitrary representations
• Hard-coded assumptions don’t apply
  – No mapping, no priority
Integration Observations

• Blocking relations between computations are independent of semantics
  – Task-2 *blocked on* Lock-1 *owned by* Task-1

• The scheduling hierarchy completely specifies system policy

• Blocking relations *in the context* of system policy have semantic relevance (e.g. PI strategy)

• *Directly representing* blocking relations *in the scheduler supports semantically independent resolution*
Solution: Directly Represent Blocking Relationships (*Proxy Execution*)

**System Policy**
- S1
- S2
- S3

**Tasks**
- Task 1
- Task 2
- Task 3
- Task 4
- Task 5

**Blocking Relations**
- Query blocking relations at scheduling time
- Semaphores publish relations

**Blocking Relations**
- T1 → L1 → T3 → L3 → T5
- T2 → L2 → T4

SRL Research Symposium, 21 October 2009
Proxy Execution Challenges

• Complexity in time and space
  – Efficient maintenance/representation of blocking relations

• Scheduler requirements
  – Scalable schedulers use set of relations indirectly

• SMP challenges
  – Relations that span CPUs require special treatment
Evaluation

• It’s difficult to prove a negative
  – Is the solution general (enough)?

• What type of wild semantics can we implement in the framework?

• Performance implications
  – For another talk
Some Results

• Static-priority, CFS, EDF

• Generalized event-based data-flow
  – Scheduler is aware of socket-based event delivery
  – PTIDES

• Guided execution
  – Deterministic execution for reproducible CC testing
    • Lock-step scheduling plans

• Application-specific progress-based scheduling
  – Multiple balanced pipelines
Conclusion

- Continually looking for interesting semantics to implement
- Currently implemented in 2.6.29-rtX

Questions?