

Scheduling

Advanced Operating Systems
(263-3800-00)

Timothy Roscoe

Thursday 16 December 2010

Scheduling is...



Deciding how to allocate a single resource among multiple clients

- In **what order** and for **how long**
 - Usually refers to CPU scheduling
 - Focus of this talk – we will look at selected systems/research
 - OS also schedules other resources (eg. disk and network IO)
 - CPU scheduling involves deciding:
 - Which task next on a given CPU?
 - For how long should a given task run?
 - On which processor should a task run?
- Task:** process, thread, domain, dispatcher, ...

Challenge: complexity of scheduling algorithms

- Scheduler also uses the CPU to decide what to schedule
 - Any time spent in scheduler is “wasted” time
 - Want to minimize overhead of decisions
 - To maximise utilization of CPU
 - But low overhead is no good if your scheduler picks the “wrong” things to run!

⇒ Trade-off between:

scheduler complexity/overhead and
optimality of resulting schedule

Challenge: Frequency of scheduling decisions

- Increased scheduling frequency
⇒ increasing chance of running something different

Leads to higher **context switching** rates,
⇒ lower throughput

- Flush pipeline, reload register state
- Maybe flush TLB, caches
- Reduces locality (eg. in cache)

L4 Scheduling

- Relatively simple scheduling policy:
 - 256 hard priority levels
 - Round robin within each priority
- Leads to simple scheduler implementation:
 - Keep (circular) list for each priority level
 - Always pick next thread from highest priority list
 - Optimization: bitmap of priority levels in-use
- L4 does lazy scheduling: avoids invoking scheduler on IPC
 - IPC send always runs receiver
 - IPC receive tries to run sender (varies, see reference manual)

L4 Scheduling: Implications



- Low overhead for scheduling, but little control over schedule
 - Can implement more complex scheduler at user-level
 - Extremely expensive, due to extra context switches
- ⇒ Fundamental problem with (CPU) scheduling:
- What policy should the kernel scheduler provide?**

Classic UNIX Scheduling



- UNIX uses multiple priorities (“nice levels”) with ageing
 - if a task has not run for a long time, its priority is increased
 - conversely, long-running tasks have their priority decreased
- Motivation: give CPU preference to IO-bound tasks
 - Improves interactivity
 - Avoids starvation
 - Hard for any one task to get 100% of the CPU
 - No ability to control/guarantee how much CPU a task will get

Problems with UNIX Scheduling



- UNIX conflates **protection domain** and **resource principal**
 - Priorities and scheduling decisions are per-process
- However, may want to allocate resources across processes, or separate resource allocation within a process
 - Eg. web server structure
 - Multi-process
 - Multi-threaded
 - Event-driven
 - If I run more compiler jobs than you, I get more CPU time
- In-kernel processing is accounted to nobody (cf. LRP)

Resource Containers

[Banga et al., 1999]



New OS abstraction for explicit resource management,
separate from process structure

- Operations to create/destroy, manage hierarchy, and associate threads or sockets with containers
- Independent of scheduling algorithms used
- All kernel operations and resource usage accounted to a resource container

⇒ Explicit and fine-grained control over resource usage

⇒ Protects against some forms of DoS attack

Lottery Scheduling

[Waldspurger and Weihl, 1994]

- Each task holds a number of “lottery tickets”
- Scheduler selects a random number, runs task holding winning ticket
- **Ticket transfers** on blocking RPC (cf. L4 donation)
- **Ticket currencies** allow local management of allocations (relative to base)
- **Compensation tickets** granted to tasks that do not use their full share (cf. UNIX ageing)
- Efficient implementation of **proportional share** resource management
- Allows policies such as “users A and B share the CPU equally”
- Hard to give guarantees

Real-Time Systems

A very brief introduction

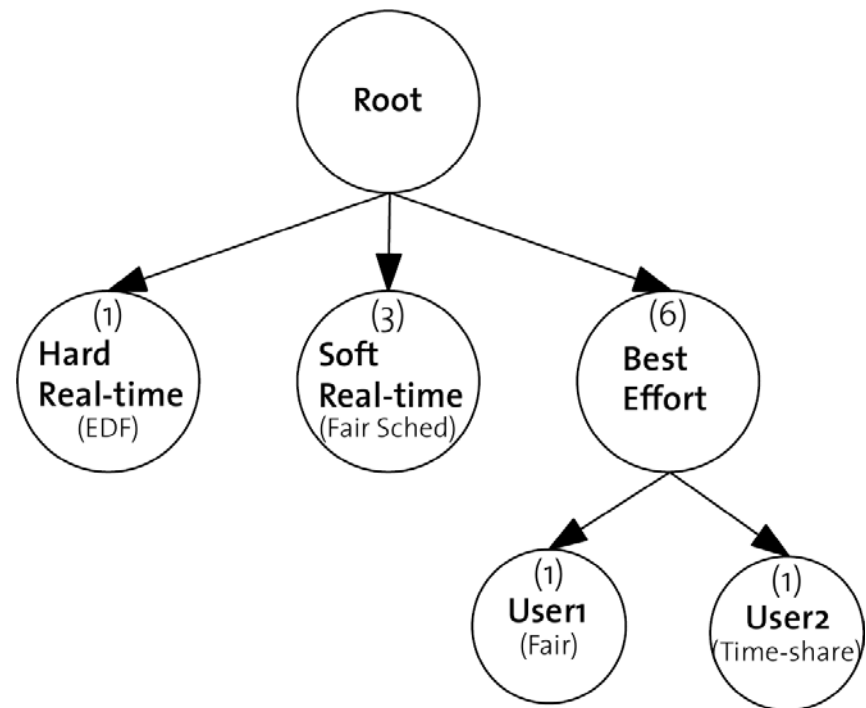


- A system where responses to external inputs must occur within a finite and specified period for correctness
- Hard vs. soft-real time
- Schedulers in commodity OSes are bad at this
 - Designed for fairness and starvation-freedom
 - Usually quantitative guarantees on CPU time
 - “Real-time” priority in Linux etc. is *not* a real time!
- Search for a “universal” CPU scheduler that integrates **real-time** and **general-purpose** requirements

Hierarchical Scheduling

[Goyal et al., 1996]

- Each task belongs to one leaf node
- Each leaf has a weight & scheduler
- Nodes scheduled hierarchically
- **SFQ** used to schedule non-leaf nodes



Start-time Fair Queueing (SFQ)

[Goyal et al., 1996]



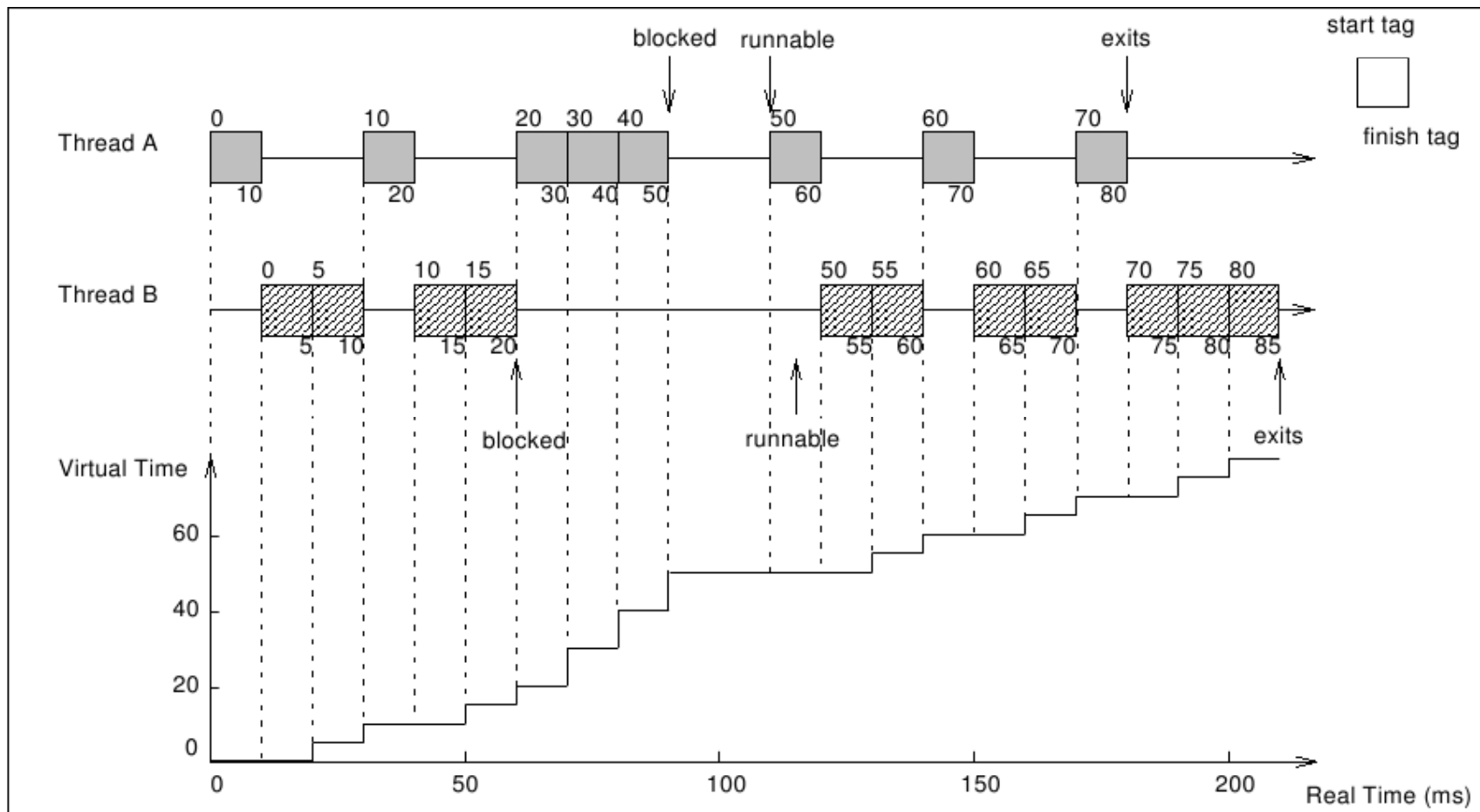
- Tasks are serviced in increasing order of start tags
- Start tag S_f and finish tag F_f defined as:

$$S_f = \max\{v(a(q_f^j)), F_j\} \quad \text{and} \quad F_f = S_f + \frac{l_f^j}{r_f}$$

- Where:
 - q_f^j = j 'th quantum of task f
 - l_f^j = length of q_f^j
 - $A(q_f^j)$: time at which the j th quantum is requested
 - r_f : weight of task f
- Virtual time $v(t)$: start tag of the task in service at time t

SFQ Example

[Goyal et al., 1996, page 5]



SFQ Summary

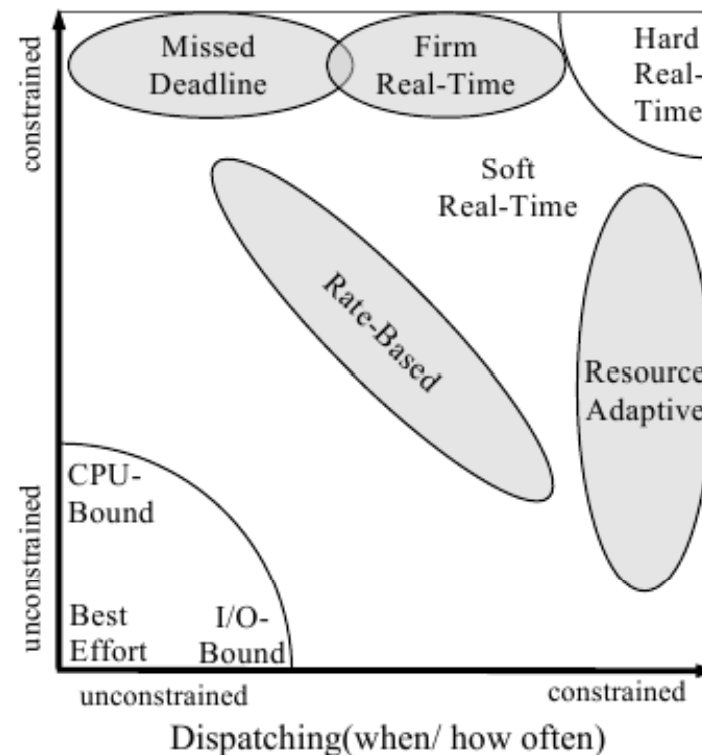
- SFQ provides hierarchical partitioning
 - Suitable for soft real-time video applications
- Hierarchical CPU scheduling claimed to:
 - Enable co-existence of heterogeneous schedulers
 - Protect application classes from each other
 - Impose similar overhead to conventional time-sharing schedulers

RAD: Resource Allocation and Dispatching

[Brandt et al., 2003]

- Resource allocation:
 - **How much** resource to allocate to each task
- Dispatching:
 - **When** to give a task the resources it has been allocated
- Most schedulers integrate management of these

(a) RAD model and real-time classes
Figure 1. Real



Rate-Based Earliest Deadline (RBED)

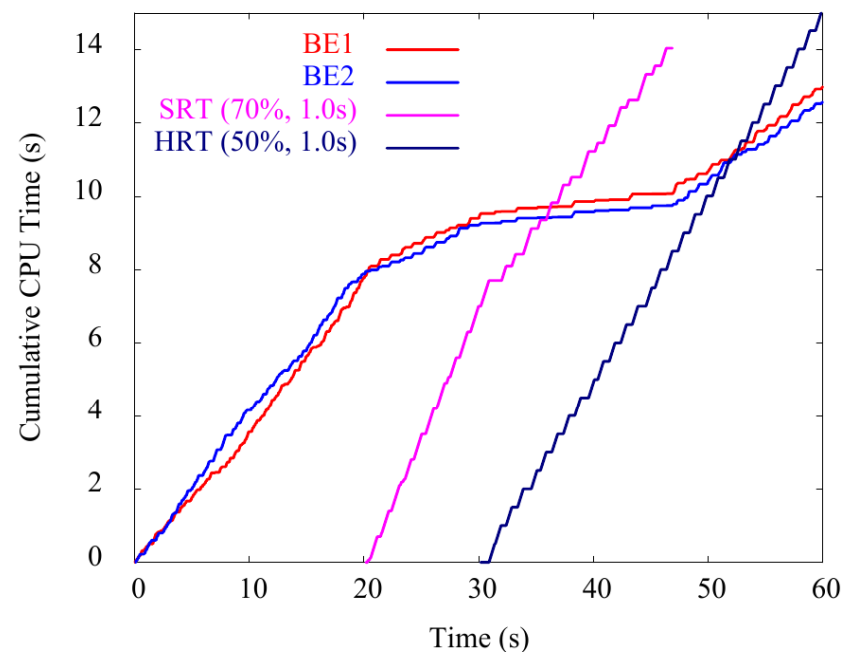
[Brandt et al., 2003]



- Earliest deadline first (EDF): run task with nearest deadline
 - How to determine “deadlines” for a non-real-time system?
- RBED policy:
 - Resource allocation: target **rate** of progress for each process
 - Dispatching: **period** based on process timeliness needs
- RBED mechanism:
 - Rate-enforcing EDF
 - Timer set for period of each task (no regular ticker!)
- **Slack** (idle/unallocated time) management important for soft real-time and background tasks
- Integrated scheduling of hard real-time, soft real-time, rate-based and best-effort tasks

RBED Results

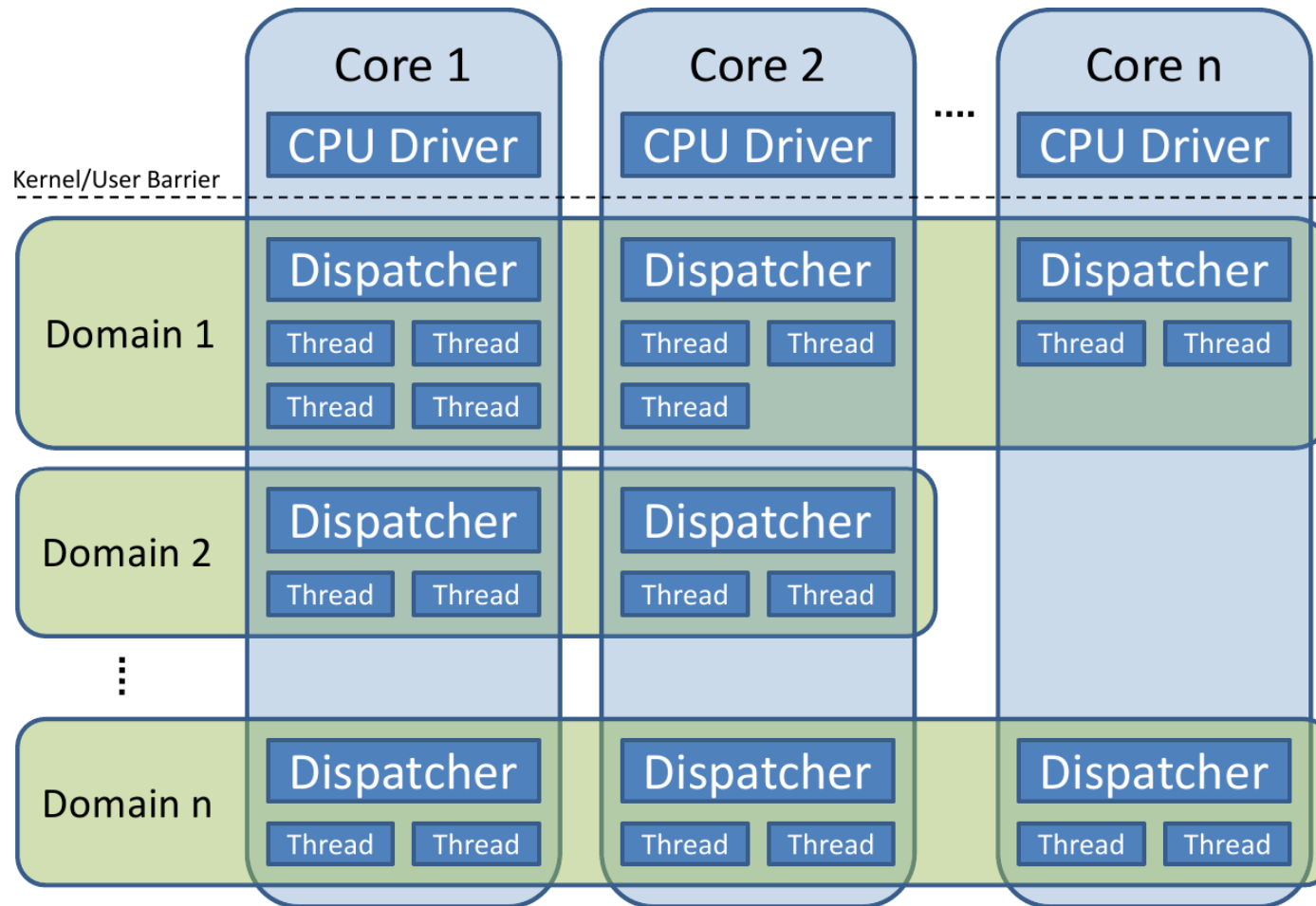
- HRT performance guaranteed by EDF
 - RBED guarantees isolation
 - Admission control resource availability
- SRT performance based on resource availability
 - Rates & periods vary when HRT or SRT task enters or exits
- Best effort tasks get whatever is left



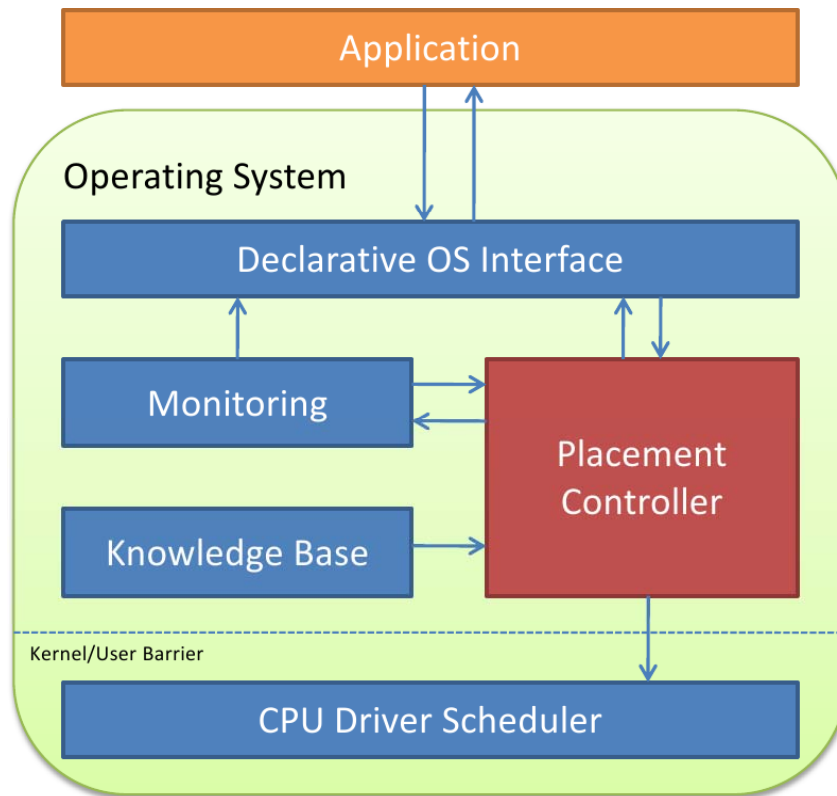
Scheduling in Barrelfish

- Biggest problem is diversity of cores, workloads
- One size doesn't fit all:
need to concurrently support different schedulers
- Placement problem: which cores run which workloads with which schedulers / scheduler parameters?

Recap: Dispatch in Barrelfish



Scheduling in Barrelfish



- Which scheduler for the CPU driver?
 - Currently RBED
- Design of placement controller open
- How to coordinate co-scheduling?

Summary

- Priorities are cheap to implement and easy to understand
 - Poor sharing behavior, poor control, unsuitable for real-time
- Real-time is now a real problem for general-purpose operating systems, due to changing workloads
- Lottery scheduling gives flexible proportional sharing
 - Doesn't (heavily) consider real-time issues
- Contrast weighted fair-queueing (SFQ, BVT) with reservation-based (classic RT, RBED) schedulers for real-time
 - Haven't discussed scheduling of other resources (eg. disk/network IO)

References



- Banga, G., Druschel, P., and Mogul, J. C. (1999). **Resource containers: A new facility for resource management in server systems**. In 3rd OSDI, pages 45–58.
- Brandt, S. A., Banachowski, S., Lin, C., and Bisson, T. (2003). **Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes**. In 24th RTSS, pages 396–409.
- Duda, K. J. and Cheriton, D. R. (1999). **Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler**. In 17th SOSP, pages 261–276.
- C. L. Liu and James W. Layland. 1973. **Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment**. J. ACM 20, 1 (January 1973), pages 46-61.
- Goyal, P., Guo, X., and Vin, H. M. (1996). **A hierarchical CPU scheduler for multimedia operating systems**. In 2nd OSDI, pages 107–122.
- Waldspurger, C. A. and Weihl, W. E. (1994). **Lottery scheduling: Flexible proportional-share resource management**. In 1st OSDI, pages 1–11.

Course wrap-up: The Exam



- Form: 15 minute oral examination
- Date: TBA (but likely in first two weeks of February)
- Location: Mothy's office CAB F 79 (not IFW!)
- Content:
 - Material from the lectures
 - Concepts you should have learned in the labs
- Focus:
 - Principles and concepts
 - Not implementation details, dates, or personalities

Concluding remarks



- Thanks for participating, and congratulations!
- If you want more, Barrelfish masters theses are available:
 - OS support for heterogeneous cores
 - Scalable storage layer
 - {Haskell,JVM,. . . } on Barrelfish
 - Multi-core resource management
 - Multi-core performance analysis and tracing
 - Online performance modelling of parallel applications
 - Making device drivers safer
 - ...

See <http://www.systems.ethz.ch/education/theses> or ask us