University of Glasgow | School of Computing Science

Wrap-up and Conclusions

Advanced Operating Systems
Lecture 16

Lecture Outline

- Review of material
- Key points
- Discussion
- Examination

Real-time Scheduling of Periodic Tasks

- Introduction and system model
  - Tasks, jobs, processors, resources
  - Timing constraints and scheduling algorithms
  - Periodic, aperiodic, and sporadic tasks
  - Hard and soft real-time systems
- Scheduling periodic tasks
  - Types of scheduler: clock-driven vs. priority-driven
  - Scheduling algorithms; approaches to proving correctness
  - Rate monotonic: non-optimality, time-demand analysis & critical instants, simply periodic systems, maximum utilisation tests
  - Earliest deadline first: optimality, maximum utilisation test, density test
  - Choice of rate monotonic vs. earliest deadline first
  - Other algorithms: deadline monotonic and least slack time

Scheduling Aperiodic and Sporadic Tasks

- Aperiodic and sporadic tasks; acceptance tests
- Scheduling aperiodic jobs
  - Background execution
  - Periodic servers: polling, deferrable, and sporadic
  - Critical instant analysis for fixed-priority deferrable server; maximum utilisation test for deferrable server in EDF systems
  - Sporadic server budget consumption/replenishment; proofs of correctness
- Scheduling sporadic jobs
  - Acceptance test in EDF systems: density of intervals
  - Acceptance test in rate monotonic systems: maximum usage over periods
- Implementation choices

Resource Management

- Resource management protocols
  - Priority inheritance protocol – simple, but transitive blocking and potential deadlock
  - Priority ceiling protocol – reduced blocking and no transitive blocking, but requires a-priori knowledge of resource usage; must track system priority ceiling; avoidance blocking prevents deadlock
  - Stack-based priority ceiling protocol – further reduction in blocking if jobs never self-suspend; blocks jobs from starting until resources available
  - Maximum duration of blocking; operation in dynamic priority systems

Programming Real-time Systems

- Real-time and embedded systems programming
  - Ensuring predictable timing
  - Device drivers – hardware interactions; options for improving robustness
  - System longevity; desire to improve robustness through alternate system implementation techniques

Garbage Collection

- Automatic memory management
  - Stack allocation
- Reference counting
  - Simple, incremental, problems with cycles
- Garbage collection
  - Mark-sweep
  - Mark-compact
  - Copying collectors
  - Generational collectors
  - Real-time collectors
- Practical factors

Message Passing

- Implications of multicore systems
  - Hardware trends; NUMA and heterogeneity in multicore systems
  - Challenges of NUMA systems – is a shared memory model appropriate?
  - Multi-kernel systems – distributed operating systems for multicore
- Message passing systems
  - Limitations of threads and lock-based concurrency
  - Multicore memory models; composition of lock-based code
  - Concepts of message passing systems
    - Interaction models; communication and the type system; naming communications
    - Message handling; immutability; linear types; use of an exchange heap
    - Pattern matching and state machines
    - Error handling; let-it-crash philosophy; supervision hierarchies; case study
  - Erlang and Scala+Akka as examples

Transactions

- Concepts of transactions
  - ACID properties
  - Concurrent execution
  - Possible to compose transactions
- Implementation challenges
  - Controlling I/O operations
  - Controlling memory access – rollback and recovery
  - Implementation using monadic concepts
- Integration into Haskell
- Integration challenges for other languages

General Purpose GPU Programming

- Heterogeneous instruction set systems
- Heterogeneous multi-kernel systems – Helios
- Main core with heterogenous offload
  - Graphics offload hardware – GPGPU
  - Programming model
  - OpenCL
  - Integration with operating systems
- Heterogenous virtual machines – Hera JVM
- Hybrid models – Accelerator
  - Lazy encoding of SIMD-style operations and JIT compilation into type system

Key Points

- Real-time systems – predictability and reliability are critical; desire to raise level of abstraction to help to achieve these goals
- Garbage collection is effective, but at high memory overhead cost – real-time garbage collection exists
- Message passing effective for multi-core systems; potential of multi-kernel operating systems model
- Transactions seem to have limited applicability
- No effective GPGPU programming model; OpenCL is too low-level and not a long-term solution

Discussion

- Wide spectrum of research ideas and concepts
- Which are seeing widespread use?
  - Functional languages and message passing concurrency
  - Garbage collection – potential for integration with kernels
  - Increased use of static code analysis tools, to debug the limitations of C
- Opportunities for dependable kernels
  - New implementation frameworks and safe programming languages
  - Approaches similar to Singularity have large potential

Examination

- Final examination:
  - Worth 80% of marks for the course
  - 2 hours; answer 3-out-of-4 questions
  - Sample exam and past papers available on Moodle, and on the website
- All material covered in the lectures, tutorials, and papers is examinable
  - Aim is to test your understanding of the material, not simply to test your memory of all the details – in particular, read papers to understand the concepts, not details
  - Explain why, don't just recite what – are looking for your reasoned and justified technical opinion about the material

The End

http://csperkins.org/teaching/adv-os/