University of Glasgow | School of Computing Science

Introduction and Course Outline

Advanced Operating Systems (M)
Lecture 1

Lecture Outline

- Administration
  - Resources
  - Aims, rationale, intended learning outcomes
  - Timetable
  - Assessment and examination
- Introduction and course outline

Resources and Contact Details

- Copies of lecture slides and other materials can be found on Moodle
  - Also at http://csperkins.org/teaching/adv-os/
  - Printed lecture handouts will not be provided – learning is enhanced by taking your own notes during lectures and tutorials

- Lecturer: Dr Colin Perkins
  - Room 405, Sir Alwyn Williams Building
  - Email: colin.perkins@glasgow.ac.uk
  - Happy to discuss the course outside timetabled hours, provided appointments are made in advance by email

Rationale

- The computing landscape has changed radically in the last decade. The desktop personal computer has become largely irrelevant and heterogeneous, multicore, mobile, and real-time systems – smart mobile phones, net books, and laptops – are now ubiquitous.
- Despite this shift, these systems are still programmed in C, and the majority run some variant of the Unix operating system.
- This course will review research on systems programming techniques and operating systems design, discuss the limitations of deployed systems, and show how the operating system infrastructure might evolve to address the challenges of supporting modern computing systems.

Aims and Objectives

- This course aims to explore the programming language and operating systems facilities essential to the implementation of real-time, reactive, and embedded systems.
- To discuss the limitations of industry-standard operating systems, and introduce new approaches to operating systems design that address the challenges of security, robustness, and concurrency.
- To give participants an understanding of the practical engineering issues caused by the design of real-time and concurrent systems; and to suggest appropriate implementation techniques for such systems.

Intended Learning Outcomes (1)

- At the end of this course, you should be able to:
  - clearly differentiate the issues that arise in designing real-time systems; analyse a variety of real-time scheduling techniques, prove correctness of the resulting schedule; implement basic scheduling algorithms;
  - apply real-time scheduling theory to the design and implementation of a real-world system using the POSIX real-time extensions; demonstrate how to manage resource access in such a system;
  - describe how embedded systems are constructed, and discuss the limitations and advantages of C as a systems programming language; understand how managed code and advanced type systems might be used in the design and implementation of future operating systems;
  - discuss the advantages and disadvantages of integrating garbage collection with the operating system/runtime; understand the operation of popular garbage collection algorithms; know when it might be appropriate to apply garbage collection and managed runtimes to real-time systems;
    …

Intended Learning Outcomes (2)

…

- understand the impact of heterogeneous multicore systems on operating systems; compare and evaluate different programming models for concurrent systems, their implementation, and their impact on operating systems;
- construct simple concurrent programs using transactional memory and message passing to understand trade-offs and implementation decisions.

Pre- and co-requisites

- Required pre-requisites:
  - Computer Systems 2
  - Operating Systems 3
  - Advanced Programming 3
  - Functional Programming 4
- Recommended co-requisites:
  - Computer Architecture 4

Course Outline

- Real-time Operating Systems
  - Clock- and priority-driven scheduling
  - Resource access control
  - Implementation techniques
- Systems Programming
- Dependable Kernels and Device Drivers
- Garbage Collection
- Concurrency
  - Transactional memory
  - Actors and Message Passing

Timetable (1)

Week	Lecture	Subject
1	Lecture 1	Introduction and Course Outline
	Lecture 2	Introduction to Real-Time Systems
	Lecture 3	Clock-Driven Scheduling of Real-time Tasks
2	Tutorial 1	Real-time Scheduling (1)
	Lecture 4	Priority-driven Scheduling of Periodic Real-time Tasks (1)
	Lecture 5	Priority-driven Scheduling of Periodic Real-time Tasks (2)
3	Tutorial 2	Real-time Scheduling (2)
	Lecture 6	Priority-driven Scheduling of Aperiodic Real-time Tasks
	Lecture 7	Priority-driven Scheduling of Sporadic Real-time Tasks
4	Tutorial 3	Real-time Scheduling (3)
	Lecture 8	Resource Access Control in Real-time Systems
	Lecture 9	Implementing Real-time Systems
5	Lecture 10	Programming Real-time and Embedded Systems
	Lecture 11	Evolution of Systems Programming
	Tutorial 4	Systems Programming

Timetable (2)

Week	Lecture	Subject
6	Lecture 12	Dependable Device Drivers
	Lecture 13	Dependable Operating Systems Architectures
	Tutorial 5	Dependable Kernels and Device Drivers
7	Lecture 14	Garbage Collection (1)
	Lecture 15	Garbage Collection (2)
	Tutorial 6	Garbage Collection
8	Lecture 16	Concurrency: Abstractions and Concepts (1)
	Lecture 17	Concurrency: Abstractions and Concepts (2)
	Tutorial 7	Concurrency (1)
9	Lecture 18	Concurrency: Transactional Memory
	Lecture 19	Concurrency: Actors and Message Passing
	Tutorial 8	Concurrency (2)
10	Lecture 20	Wrap-up and Review

Assessment

- Level M course, worth 10 credits
- Coursework (20%)
  - 5% real-time scheduling: periodic tasks (set lecture 5; due Monday wk 4)
  - 5% real-time scheduling: aperiodic tasks (set lecture 7; due Monday wk 5)
  - 10% essay: OS kernel evolution (set tutorial 5; due Monday wk 9)
- Examination (80%)
  - Duration 2 hours; sample paper will be made available on Moodle
  - All material in the lectures, tutorials, and background reading is examinable
  - Aim is to test your understanding of the material, not to test your memory of all the details; explain why – don't just recite what

Required Reading

- No set textbook, but research papers will be cited
  - DOIs will be provided: resolve via http://dx.doi.org/
  - You are expected to read and understand these; it will be beneficial to follow-up on some of the references and do further background reading
  - Tutorials allow for discussion of papers and lectured material
- If you're not used to reading research papers, learn how to do so
  - Critical reading of a research paper is a difficult skill, and requires practice. Read in a structured manner, not end-to-end, thinking about the material as you go. You'll need to take notes as you read, and go through the paper more than once.
  - http://www.eecs.harvard.edu/~michaelm/postscripts/ReadPaper.pdf
  - S. Keshav, "How to Read a Paper", ACM CCR, 37(3), DOI 10.1145/1273445.1273458

Advanced Operating Systems

- Unix/Linux and Windows are the outcome of a long strand of operating systems development
  - The C programming language
  - Monolithic kernels
    - Unix – unbroken line of evolution since the early 1970s
    - Windows – builds on Digital Equipment Corporation VAX/VMS dating from 1975

- Operating systems and programming language research have evolved since the 1970s – how might this affect future operating systems?

Real-time Operating Systems

- Introduction to real-time systems
- Real-time scheduling
  - Clock driven scheduling
  - Priority driven scheduling:
    - Periodic, aperiodic and sporadic tasks
    - Rate and deadline monotonic scheduling, earliest deadline first, least slack time
  - Proofs of correctness
- Resource access control
  - Priority inheritance protocol; priority ceiling protocol
  - Impact of scheduling
- Implementation techniques
  - Real-time APIs and code; implementing real-time schedulers

Systems Programming

- Programming real-time and embedded systems
  - Interacting with hardware
  - Interrupt and timer latency
  - Memory issues
  - Power, size and performance constraints
- System longevity
- Development and debugging

- Traditional approaches; possible future alternatives
  - Moving beyond C for the embedded world

Dependable Device Drivers

- Sources of bugs in device drivers
- Engineering approaches to improving device driver reliability
  - Use of object-oriented code and languages for device drivers
  - MacOS X I/O Kit as a example
- Future directions: explicit identification of driver state machines
  - Formal verification driver code
  - Integration with model checking
  - Dingo and Singularity as examples

Dependable Kernels

- Evolution of the operating system kernel
  - Microkernels
  - Use of managed code for systems programming – how much of the kernel can be written in a high-level type-safe language?
  - Pervasive concurrency
  - Examples: Singularity and BarrelFish

Garbage Collection

- Memory management models
  - Garbage collection – advantages and disadvantages
  - Other approaches
- Role of garbage collection in future operating systems
- Garbage collection algorithms and their properties
- Real-time garbage collection

Concurrency

- Pervasive concurrency, and its implications for next generation operating systems
- Software Transactional Memory
  - Transactional processing as the fundamental concurrency primitive
  - Relation to purely functional languages
  - Implementation in Haskell
- Actors and message passing
  - Exchange of immutable messages between concurrent processes as the fundamental concurrency primitive
  - Implications for locking
  - Linear types
  - Implementation in Erlang and Singularity

Summary and Next Steps

- Real-time Operating Systems
  - Clock- and priority-driven scheduling
  - Resource access control
  - Implementation techniques
- Systems Programming
- Dependable Device Drivers
- Dependable Kernels
- Garbage Collection
- Concurrency
  - Transactional memory
  - Actors and Message Passing

- Next lecture – begins discussion of real-time systems