Advanced Operating Systems
Through tracing, analysis, and experimentation

ACS/Part III L41: Advanced Operating Systems
Part II: Advanced Operating Systems

Lecture 1, Part 2: The Course

Dr Robert N. M. Watson

2020-2021

Why study operating systems?

The OS plays a central role in **whole-system design** when building efficient, effective, and secure systems:

- Strong influence on whole-system performance
- Critical foundation for computer security
- Exciting programming techniques, algorithms, problems
  - Virtual memory; network stack; filesystem; run-time linker; …
- Co-evolves with platforms, applications, users
- Multiple active research communities
- Reusable techniques for building complex systems
- Boatloads of fun (best text adventure ever)

Where is the OS research?

A sub-genre of **systems research**:

- Evolving hardware-software interfaces
  - New computation models/architectures
  - New kinds of peripheral devices
- Integration with programming languages and runtimes
- Concurrent/parallel programming models; scheduling
- Security and virtualisation
- Networking, storage, and distributed systems
- Tracing and debugging techniques
- Formal modeling and verification
- As a platform for other research – e.g., mobile systems

Venues: SOSP, OSDI; ATC; EuroSys; HotOS; FAST; NSDI; HotNets; ASPLOS; USENIX Sec.; ACM CCS; IEEE SSP; …

What are the research questions?

Just a few examples: By changing the OS, can I…

- Create new abstractions for new hardware?
- Make my application run faster by…
  - Better masking latency?
  - Using parallelism more effectively?
  - Exploiting new storage mediums?
  - Adopting distributed-system ideas in local systems?
- Make my application more {reliable, energy efficient}
- Limit {security, privacy} impact of exploited programs?
- Use new language/analysis techniques in new ways?

Systems research focuses on **evaluation** with respect to **applications** or **workloads**: How can we measure whether it is {faster, better, ...}?

Teaching operating systems

- Two common teaching tropes:
  - **Trial by fire**: in micro, recreate classic elements of operating systems: microkernels with processes, filesystems, etc.
  - **Research readings course**: read, present, discuss, and write about classic works in systems research
- This module adopts elements of both styles while:
  - mitigating the risk of OS kernel hacking in a short course
  - working on real-world systems rather than toys; and
  - targeting research skills not just operating-system design
- Trace and analyse real systems driven by specially crafted benchmarks
- Possible only because of (fairly) recent developments in tracing and hardware-based performance analysis tools

Aims of the module (1/2)

Teaching **methodology**, **skills**, and **knowledge** required to understand and perform research on contemporary operating systems by…

- Employing systems methodology and practice
- Exploring real-world systems artefacts through performance and functional evaluation/analysis
- Developing scientific writing skills **(L41 only)**
- Reading original systems research **(L41 only)**

Aims of the module (2/2)

On completion of this module, students should:

- Have an understanding of high-level OS kernel structure.
- Gained insight into hardware-software interactions for compute and I/O.
- Have practical skills in system tracing and performance analysis.
- Have been exposed to research ideas in system structure and behaviour. **(L41 only)**
- Have learned how to write systems-style performance evaluations. **(L41 only)**

Prerequisites

We will take for granted:

- **High-level knowledge of OS terminology** from an undergraduate course (or equivalent); e.g.,:
  - What **schedulers** do
  - What **processes** are … and how they differ from threads
  - What **Inter-Process Communication (IPC**) does
  - How might a simple **filesystem** might work
- Reasonable fluency in **reading** multithreaded C
- Good working knowledge of Python
- Comfort with the UNIX command-line environment
- Undergraduate skills with statistics
  (mean/median/mode/stddev/*t*-tests/linear regression/boxplots/scatterplots ...)

You can pick up some of this as you go (e.g., IPC, Python, or *t*-tests), but will struggle if you are missing several

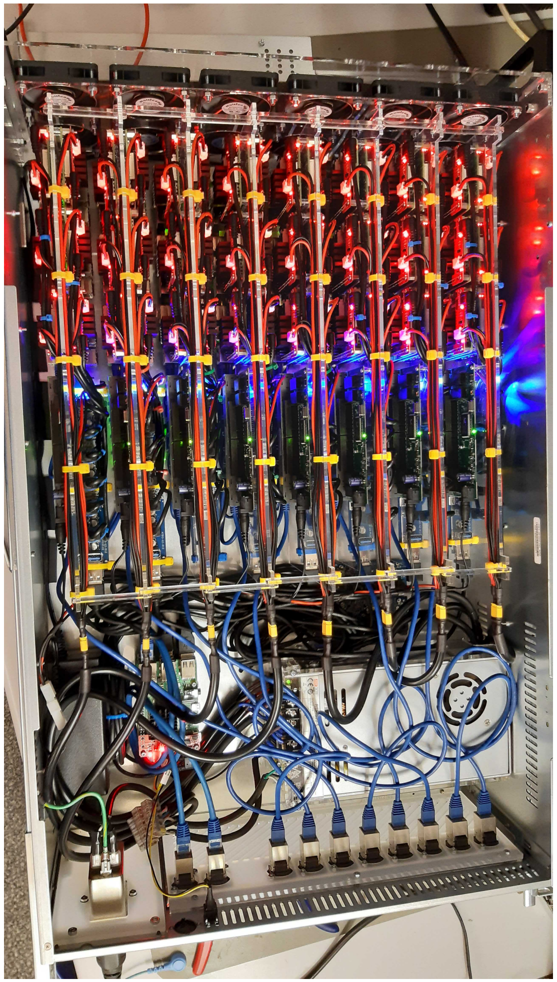
Module structure – four complementary strands

- **6 hours of baseline lecture content online**
  - Theory, methodology, architecture, and practice
- **5 sets of lab exercises online (2 optional for Part II)**
  - Short recorded lecturelet introduces the lab
  - Online remote access RPi4 cluster to run experiments
  - Module demonstrators available online to ask questions
- **Assigned research and applied readings**
  - Selected portions of module texts – learn skills, methodology
  - Related research readings – research exposure **(L41 only)**
- **Marked lab assignments (AdvOpSys) or reports (L41)**
  - Based on experiments done via lab exercises
  - Develop scientific + writing skills suitable for systems research
  - First lab has lower weighting as you learn baseline skills

Outline of module schedule

- Submodule 1: **Introduction to kernels and tracing/analysis**
  - 1 lecture, 1 lab (I/O)
  - **Introduction**: OSes, Systems Research, and L41
  - **The Kernel**: Kernel and Tracing
- Submodule 2: **The Process Model**
  - 1 lecture, 2 labs (IPC, PMC)
  - **The Process Model** (1) – Binaries and Processes
  - **The Process Model** (2) – Traps, System Calls, and Virtual Memory
- Submodule 3: **The Network Stack (TCP/IP)**
  - 1 lecture, 2 labs (TCP state machine, congestion control)
  - **The Network Stack** (1) – Sockets, NICs, and Work Distribution
  - **The Network Stack** (2) – TCP protocol
- Each submodule has one lab report or lab assignment
- Please consult online materials for all deadlines

The lab platform



- 50x Raspberry Pi 4 boards in a rack
  - Broadcom BCM2711 SoC
  - 4x 64-bit A72 ARMv8-A cores
  - 8GB DRAM, 64G SD Card
- FreeBSD 13-CURRENT (development branch)
  - DTrace tracing tool
  - HWPMC counter framework
  - Bespoke potted benchmarks motivating OS and microarchitectural performance analysis
  - Jupyter lab notebook environment
- Remotely accessed via SSH + tunneling for Jupyter

Lab Assignments **(Part II only)**

- A series of question-and-answer assignments
  - Perform your work in the Jupyter lab framework
  - Collect, analyse, and present performance results
  - Your submission will consist of generated PDF of the completed lab notebook
  - Submissions only via Moodle
  - Your mark is based on submitted data, text, and plots

- **2020-2021 course only**
  - The third lab assignment (TCP/IP networking) is optional

Lab Reports **(L41 only)**

Lab reports document an experiment and analyse its results – typically using **one or more hypotheses**.

Our lab reports will contain the following sections (see notes, template):

1. Title + abstract (1 page)	5. Conclusion (1-2 para)
2. Introduction (1-2 para)	6. References
3. Experimental setup and methodology (1-2 pages)	7. Appendices
4. Results and discussion (3-4 pages)	

Some formats break out (e.g.) experimental setup vs. methodology, and results vs. discussion. The combined format seems to work better for systems experimentation as compared to (e.g.) biology.

- The target length is **8 pages excluding appendices, references**
- **Over-length reports** will be penalized – please stop by the limit!
- **Appendices** will not be read if too long, and should not be essential to understanding the core content of the report

Module texts – core material

You will need to make frequent reference to these books both in the labs and outside of the classroom:

Operating systems: Marshall Kirk McKusick, George V. Neville-Neil, and Robert N. M. Watson, ***The Design and Implementation of the FreeBSD Operating System, 2nd Edition***, Pearson Education, Boston, MA, USA, September 2014.

Performance measurement: Raj Jain, ***The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling***, Wiley - Interscience, New York, NY, USA, April 1991.

Tracing and profiling: Brendan Gregg and Jim Mauro, ***DTrace: Dynamic Tracing in Oracle Solaris***, Mac OS X and FreeBSD, Prentice Hall Press, Upper Saddle River, NJ, USA, April 2011.

The FreeBSD and DTrace books are available online via vlebooks.com:

https://www.vlebooks.com/Vleweb/Search/Keyword?keyword=freebsd

Module texts – additional material

If your OS recollections feel a bit hazy:

Operating systems: Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. ***Operating System Concepts***, Eighth Edition, John Wiley & Sons, Inc., New York, NY, USA, July 2008.

If you want to learn a bit more about architecture and measurement:

Performance measurement and diagnosis: Brendan Gregg, ***Systems Performance: Enterprise and the Cloud***, Prentice Hall Press, Upper Saddle River, NJ, USA, October 2013.

Wrapping up

- In this lecture, we have:
  - Explored the idea of an operating system
  - Detailed the structure of the course and its expectations

- Our next lecture will explore:
  - DTrace, the kernel tracing facility we will use
  - The *probe effect* and its impact
  - The dynamics of kernel execution (just a taster)

- Readings for the next lecture:
  - Paper - Cantrill, et al. 2004
  - McKusick, et al. Chapter 3 (Kernel Subsystems)