Advanced Operating Systems
Through tracing, analysis, and experimentation

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

Lecture 1, Part 2: The Course
Prof. Robert N. M. Watson
2023-2024
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 (**Part III/ACS only**)
• Reading original systems research (**Part III/ACS 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. *(Part III/ACS only)*

• Have learned how to write systems-style performance evaluations. *(Part III/ACS 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

• Lectures (×5: 4 in-person 2-hour slots, 1 prerecorded)
  • Theory, methodology, architecture, and practice

• Assigned research and applied readings
  • Selected portions of module texts – learn skills, methodology
  • Related research readings – research exposure (L41 only)

• In-person lab exercises (×3 labs, prerecorded lecturelets)
  • Short prerecorded lecturelet introduces each lab
  • RPi4 cluster to run experiments (one board per student)
  • 6× Module demonstrators available to answer questions

• First lab assignment
  • Acclimate to platform
  • Learn essential skills to perform later labs (e.g., DTrace, Jupyter)

• Later lab assignments (Part II – ×2) or reports (L41 – ×2)
  • Based on experiments done in lab exercises
  • Develop scientific + writing skills suitable for systems research (L41)
Outline of module schedule

• Submodule 1: **Introduction to kernels and tracing/analysis**
  • 2 lectures (one prerecorded), 1 lab on kernel tracing
  • **Introduction**: OSes, Systems Research, and L41
  • **The Kernel**: Kernel and Tracing

• Submodule 2: **The Process Model**
  • 2 lectures, 1 lab on IPC
  • **The Process Model** (1) – Binaries and Processes
  • **The Process Model** (2) – Traps, System Calls, and Virtual Memory

• Submodule 3: **The Network Stack (TCP/IP)**
  • 2 lectures, 1 lab on TCP
  • **The Network Stack** (1) – Implementation and research
  • **The Network Stack** (2) – TCP and its implementation

• 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 operating system
  • 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
Shared first Lab 1: Getting started with kernel tracing

• Identical assignment for Part II and Part III/ACS
• Exercises to get you started on the platform; teach:
  • Jupyter Lab Notebooks
  • DTrace instrumentation and data collection – in particular, tracing and profiling scripts
  • Relevant Python plotting tools including Flame Graphs
  • And first dirty hands with respect to OS internals
• Submitted only via Moodle; use “Print to PDF” in your browser to generate a PDF to submit
• Low proportion of marks (10% for Part III/ACS; optional for Part II): really about teaching basic skills you will need for later labs
  • Experience confirms that students who don’t do the first lab will do badly on later labs; correlation .. maybe causality?
• Deadline is 12:00 on 1 December 2023
Lab Assignments 2 and 3 (Part II only)

• A series of questions requiring short answers
  • Answers consist of written text, selected data, and plots
  • Perform your work in the Jupyter lab framework
  • Your submission will consist of generated PDF of the completed lab notebook – e.g., by printing to a PDF file
  • Submissions are accepted only via Moodle

• Ensure that your submission is well presented; e.g.,
  • Plots don’t span page boundaries or run off the side
  • Plots have clearly labeled axes, data sets, and so on
  • Make sure your text is concise and clear, addressing the questions that are answered

• Marked based on submitted data, text, and plots
Lab Reports 2 and 3 (Part III/ACS only)

Lab reports document an experiment and analyse its results – typically using one or more hypotheses (which we will provide).

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

<table>
<thead>
<tr>
<th>1. Title + abstract (1 page)</th>
<th>5. Conclusion (1-2 para)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Introduction (1-2 para)</td>
<td>6. References</td>
</tr>
<tr>
<td>3. Experimental setup and methodology (1-2 pages)</td>
<td>7. Appendices</td>
</tr>
<tr>
<td>4. Results and discussion (3-4 pages)</td>
<td></td>
</tr>
</tbody>
</table>

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:


The FreeBSD and DTrace books are available online via vlebooks.com: [https://www.vlebooks.com/Vleweb/Search/Keyword?keyword=freebsd](https://www.vlebooks.com/Vleweb/Search/Keyword?keyword=freebsd)
Module texts – additional material

If your OS recollections feel a bit hazy:


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