L41: Advanced Operating Systems

Through tracing, analysis, and experimentation

L41 Lecture 1

Dr Robert N. M. Watson

23 October 2017

Getting started

- What is an operating system?
- Systems research
- About the module
- Lab reports
- Readings for next time

What is an operating system?

(Whiteboarding exercise)

What is an operating system?

[An OS is] low-level software that supports
a computer's basic functions, such as
scheduling tasks and controlling peripherals.

- Google hive mind

General-purpose operating systems

… are for **general-purpose computers:**

- Servers, workstations, mobile devices
- Run **applications** – i.e., software unknown at design time
- Abstract the hardware, provide 'class libraries'
- E.g., Windows, Mac OS X, Android, iOS, Linux, FreeBSD, …

Userspace	Local and remote shells, management tools, daemons Run-time linker, system libraries, logging and tracing facilities
	– system-call layer –
Kernel	System calls, hypercalls, remote procedure call (RPC)* Processes, filesystems, IPC, sockets, management Drivers, packets/blocks, protocols, tracing, virtualisation VM, malloc, linker, scheduler, threads, timers, tasks, locks

* Continuing disagreement on whether distributed-filesystem servers and window systems 'belong' in userspace or the kernel

Other kinds of operating systems

Specialise the OS for a specific application or environment:

- **Embedded, real-time operating systems**
  - Serve a single application in a specific context
  - E.g., WiFi access points, medical devices, washing machines, cars
  - Small code footprint, real-time scheduling
  - Might have virtual memory / process model
  - Microkernels or single-address space: VxWorks, RTEMS, L4
  - Now also: Linux, BSD (sometimes over a real-time kernel)
- **Appliance operating systems**
  - Apply embedded model to higher-level devices/applications
  - File storage appliances, routers, firewalls, …
  - E.g., Juniper JunOS, Cisco IOS, NetApp OnTap, EMC/Isilon
  - Under the hood, almost always Linux, BSD, etc.

Key concept: **Operating system as a reusable component**

Other kinds of operating systems?

What if we rearrange the boxes?

- **Microkernels, library operating systems, unikernels**
  - Shift code out of the kernel into userspace to reduce TCB; improve robustness/flexibility; 'bare-metal' apps
  - Early 1990s: Microkernels are king!
  - Late 1990s: Microkernels are too slow!
  - 2000s/2010s: Microkernels are back! But now 'hypervisors'
  - Sometimes: programming-language runtime as OS
- **Hypervisors**
  - Kernels host applications; hypervisors host virtual machines
  - Virtualised hardware interface rather than POSIX
  - Paravirtualisation reintroduces OS-like interfaces for performance
  - A lot of microkernel ideas have found a home here
  - E.g., System/370, VMware, Xen, KVM, VirtualBox, bhyve, ...
- **Containers**
  - Host OS as hypervisor, but using the process model
  - Really more about code/ABI distribution and maintenance

What does an operating system do?

- Key hardware-software surface
- Low-level abstractions and services
  - **Operational model**: bootstrap, shutdown, watchdogs
  - **Process model, IPC**: processes, threads, IPC, program model
  - **Resource sharing**: scheduling, multiplexing, virtualisation
  - **I/O**: drivers, local/distributed filesystems, network stack
  - **Security**: authentication, encryption, ACLs, MAC, audit
  - **Local or remote access**: console, window system, SSH
  - **Libraries**: math, protocols, RPC, crypto, UI, multimedia
  - **Monitoring/debugging**: logs, profiling, tracing, debugging

Compiler? Text editor? E-mail package? Web browser?
Can an operating system be "distributed"?

L41 Lecture 1 - Advanced Operating Systems

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, ...}?

(See RSP lecture on Systems Experimentation in Lent)

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 recent developments in tracing and hardware-based performance analysis tools

Aims of the module

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
- Reading selected original systems research papers

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
- Working knowledge of Python (or R)
- 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, *t*-tests), but will struggle if you are missing several

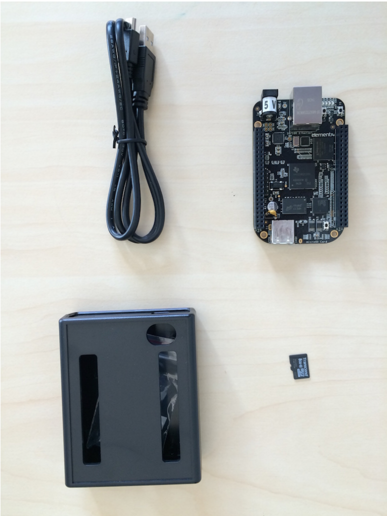
Module structure – four complementary strands

- **6x one-hour lectures** in SW-01
  - Theory, methodology, architecture, and practice
- **5x two-hour labs** in SW-02
  - Start with 10-20-minute lecturelets on artefacts, practical skills
  - Remainder on hands-on measurement and experimentation – learn skills required to write assigned lab reports, start on experiments
  - Lab **experimental questions** must be answered in your lab reports
- **Assigned research and applied readings**
  - Selected portions of module texts – learn skills, methodology
  - Historic and contemporary research papers – research exposure
- **Marked lab reports**
  - Based on experiments done in (and out) of scheduled labs
  - Refine scientific writing style suitable for systems research
  - One 'practice run' marked but not assessed ← **not optional!**
  - Two assessed; 50% of final mark each

Rough module schedule

- Submodule 1: **Introduction to kernels and tracing/analysis**
  - 2 lectures, 1 lab (I/O)
  - **Introduction**: OSes, Systems Research, and L41
  - **The Kernel**: Kernel and Tracing
  - First lab report due
- Submodule 2: **The Process Model**
  - 2 lectures, 2 labs (IPC, PMC)
  - **The Process Model** (1) – Binaries and Processes
  - **The Process Model** (2) –Traps, System Calls, and Virtual Memory
  - Second lab report due
- Submodule 3: **TCP/IP**
  - 2 lectures, 2 labs (TCP state machine, congestion control)
  - **The Network Stack** (1) – Sockets, NICs, and Work Distribution
  - **The Network Stack** (2) – TCP protocol
  - Final lab report due

The platform



- BeagleBone Black
- 1GHz ARM Cortex A-8 32-bit CPU
- Superscalar pipeline, MMU, L1/L2 caches
- FreeBSD operating system + DTrace
- "Potted benchmarks"
- iPython-based measurement and analysis environment

Labs and lab reports

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 **10 pages excluding appendices, references**
- **Over-length reports** will be assessed within page limit
- **Appendices** may 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.

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.

For next time

- McKusick, et al. – Chapter 3
- Cantrill, et al. 2004 – full article