L41: Advanced Operating Systems
Through tracing, analysis, and experimentation

Dr Robert N. M. Watson

23 February 2015
Getting started

1. What is an operating system?
2. Systems research
3. About the module
4. Lab reports
5. 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
What is an operating system?

**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, tracing facilities

--- system-call interface ---

**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
What is an operating system?

Other kinds of operating systems

What if we specialise the OS for specific applications or environments?

- **Embedded operating systems**
  - Serve a single application in a specific context
  - Wireless 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 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
What is an operating system?

Other kinds of operating systems (cont.)

What if we rearrange the process-model and system-call 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: alternative linkage (e.g., programming-language)

- **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, ...
What is an operating system?

What does an operating system do?

- Key hardware-software surface (cf. compilers)
- System management: bootstrap, shutdown, watchdogs
- Low-level abstractions and services
  - Programming: processes, threads, IPC, program model
  - Resource sharing: scheduling, multiplexing, virtualisation
  - I/O: device drivers, local/distributed filesystems, network stack
  - Security: authentication, encryption, permissions, labels, audit
  - Local or remote access: console, window system, SSH
- Libraries: math, protocols, RPC, cryptography, UI, multimedia
- Other stuff: system log, debugging, profiling, tracing

Compiler? Text editor? E-mail package? Web browser? Can an operating system be ‘distributed’?
Why study operating systems?

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

- Key interface between hardware and software
- Strong influence on whole-system performance
- Critical foundation for computer security
- Exciting programming techniques, algorithms, problems
  - Virtual memory; network stacks; filesystems; runtime linkers; ...
- Co-evolves with platforms, applications, users
- Multiple active research communities
- Reusable techniques for building complex systems
- Boatloads of fun (best text adventure ever)
Why study operating systems?

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
- As a platform for other research – e.g., mobile systems

Venues: SOSP; OSDI; USENIX ATC; EuroSys; HotOS; FAST; NSDI; HotNets; ASPLOS; USENIX Security; ACM CCS; IEEE S&P; ...
Why study operating systems?

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, power efficient}
- limit the {security, privacy} implications of bad/exploited programs?
- use new languages/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 of systems research

This module adopts elements of both approaches 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 synthetic benchmarks
- Possible only because of recent developments in tracing
Aims of the module

Teach methodology, skills, and knowledge required to understand and perform research on contemporary operating systems by...

- Teaching systems-analysis methodology and practice
- Exploring real-world systems artefacts
- Developing scientific writing skills
- Reading selected original systems research papers
Prerequisites

We will take for granted:

- High-level knowledge of operating systems gained in an undergraduate course (or equivalent); e.g.:
  - What schedulers do
  - What processes are... and how they differ from threads
  - What Interprocess Communication (IPC) does
  - How a simple filesystem might work

- Reasonable fluency in reading/producing multithreaded C

- 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, \(t\)-tests), but will struggle if you are missing most of these
Module structure

- **One-hour lectures** in FS-11 (×6)
  - Theory, methodology, architecture, and practice

- **Two-hour labs** in SW-02 (×5)
  - 10–20-minute lecturelets on artefacts and practical skills
  - Remainder on hands-on measurement and experimentation
  - Optional *exploratory questions vs. experimental goals*

- **Assigned readings**
  - Selected portions of module texts
  - Historic and contemporary research papers

- **Assessed lab reports**
  - Based on experiments done in (and out) of labs
  - Refine scientific writing style suitable for systems research
  - One ‘practice run’ marked but not assessed
  - Two assessed; 50% of final mark each
Rough module schedule and decomposition

- **Weeks 1-2: Operating systems and tracing**
  - Operating-system structures
  - Kernel tracing
  - 2 lectures, 1 lab (I/O)
  - First lab report due

- **Weeks 3-5: Processes, threads, and scheduling**
  - Trace and optimise
  - Scheduling, the hardware-software interface
  - 2 lectures, 2 labs (IPC, PMC)
  - Second lab report due

- **Weeks 6-8: Sockets and TCP**
  - Trace and optimise TCP behaviour
  - Characterising the **probe effect**
  - 2 lectures, 2 labs (TCP/sockets, congestion control)
  - 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’
Labs and lab reports

Lab reports document an experiment and analyse its results – typically in the context of a hypothesis that will be tested.

A lab report will contain at least the following sections:

1. Title + abstract  
2. Introduction  
3. Experimental setup and methodology  
4. Results and discussion  
5. Conclusion  
6. References  
7. Appendices  

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

See lab-report notes on the website.
Module texts - core material

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


Module texts - supplemental material

If your OS recollections feel a bit hazy:


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

Reading for Wednesday

- McKusick, et al - Chapters 2-3
- Cantrill, et al - full article