Advanced Operating Systems

Through tracing, analysis, and experimentation

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

Lecture 1, Part 1: What is an Operating System?

Prof. Robert N. M. Watson

2021-2022

MPhil/Part III L41 vs. Part II AdvOpSys

- These lectures are shared by two separate courses:
  - ACS / Part III L41: Advanced Operating Systems
  - Part II: Advanced Operating Systems (AdvOpSys)
- The two courses also share an online lab framework based on the RPi4, DTrace, and HWPMC

- But there are some important differences:
  - Key difference 1: Assessed coursework
    - L41 has **3x** independently written **lab reports**
    - Part II has **2x** short answer **lab assignments** (+ 1x optional)
  - Key difference 2: Assigned readings
    - L41 assigns additional research readings
- Please be sure to use the right material for your course!

Getting started

- What is an operating system?

Lecture 1, Part 1

- About the module
- Systems research
- Lab assignments / reports
- Readings for next lecture

Lecture 1, Part 2

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?

But that is basically the 1970s definition,
and not at all a contemporary one.

Today's general-purpose operating systems consist of
GB of binaries and hundreds of millions of LoC.

Further, when you select an operating system,
you select hardware and software ecosystems.

What is an operating system?

Payment services?

Access control?

Threads and processes?

Backup?

Crypto libraries?

Software updates?

Local file systems?

Debug and trace?

Networking and WiFi?

Secure enclaves?

Window system?

User authentication?

Application packaging?

Shell and command-line tools?

Distributed file-system clients and servers?

Kernel and userspace?

Web browser?

Profiling and optimization?

Device drivers?

Remote management?

Class libraries?

Crashdump collection

System libraries?

Virtual machines?

Run-time linker?

Remote access?

.. And surely lots more

Language runtimes?

Multimedia?

General-purpose operating systems

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

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

Userspace	Local and remote shells, GUI, 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 (1/3)

Specialise the OS for a specific application or environment:

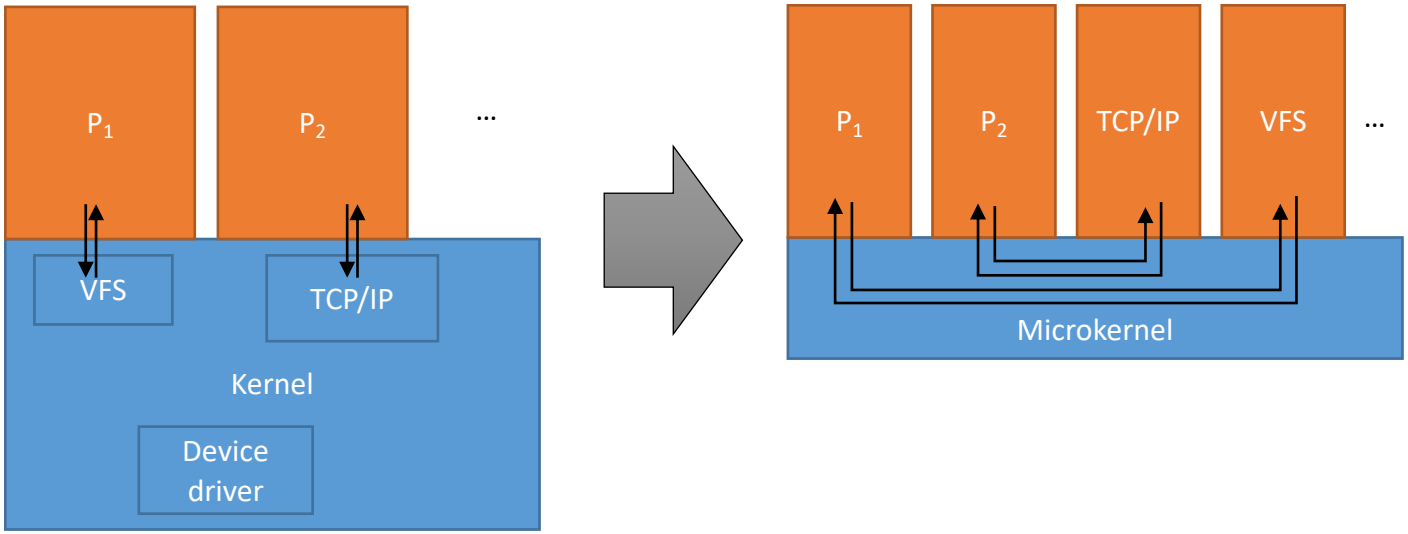
- **Embedded, real-time operating systems**
  - Serve a single application in a specific context
    - E.g., WiFi access points, medical devices, washing machines, cars
  - Small code footprint, real-time scheduling
  - Might have virtual memory / process model
  - Microkernels or single-address space: VxWorks, RTEMS, L4
  - Now also: Linux, BSD (sometimes over a real-time kernel), etc.
- **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? (2/3)

What if we rearrange the boxes?

- **Microkernels, library operating systems, unikernels**
  - Shift code from kernel into userspace to reduce Trusted Computing Base (TCB); improve robustness/flexibility; 'bare-metal' apps
  - Early 1990s: Microkernels are king!
  - Late 1990s: Microkernels are too slow!
    - (But ideas about OS modularity dating from this period are widespread)
  - 2000s/2010s: Microkernels are back! But now 'hypervisors'
  - Sometimes: programming-language runtime as OS



Other kinds of operating systems? (3/3)

- **Hypervisors**
  - Kernels host processes; hypervisors host virtual machines
    - Type-1: Standalone hypervisors (e.g., Xen)
    - Type-2: Integrated with OS kernel (e.g., KVM)
  - Virtualised hardware interface rather than POSIX APIs
  - Paravirtualisation reintroduces OS-like APIs for performance
  - E.g., System/370, VMware, Xen, KVM, VirtualBox, bhyve, Hafnium, ...
  - Many microkernel ideas have found a home here
- **Containers**
  - Hosts multiple userspace instances over a common kernel
  - Controlled namespaces prevent inappropriate accesses
  - Really more about code/ABI (Application Binary Interface) distribution and maintenance

What does an operating system do?

- Key hardware-software surface (w/compiler toolchain)
- 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”?