Advanced Operating Systems
(263-3800-00L)

Timothy Roscoe
Herbstsemester 2014
http://www.systems.ethz.ch/courses/fall2014/aos
Course aims and goals
Writing real OS code

• There’s a limit to how much you can learn about an OS by reading books.

• Solve common OS problems for yourself
  – Dealing with concurrency
  – Dealing with asynchrony
  – Tailoring code to specific hardware
  – Architecting complete system of processes
It’s not Unix

The course avoids using a Unix-like OS:

• Gives a **broader outlook** on what an OS can be
• Encourages a **critical approach** to OS designs
• Better **understanding** of the structure of Unix / Linux / Windows / Android / iOS / ...
• Clearer picture of **ideas** (finally) appearing in older OS designs
  – Or not yet...
What is OS research?

We’ll be working with a research OS

• You’ll get a sense of how OS research is done
  – Read research papers
  – Relate them to implementations
  – How to evaluate such ideas

• See what constitutes an OS research topic
  – What problems do these ideas solve?
  – How are they motivated, where do they come from?
Key ideas in OS research

• The OS we use includes a selection of key ideas in OS research
  – You get hands-on experience with the ideas
  – We’ll go through the papers on the topic
  – We’ll put them into historical context

• Many are not well-known and/or unimplemented in the wider community
Motivation
Why teach non-Unix OS design?

• This is an interesting time for OS research:
  – Multicore
  – Heterogeneous processing
  – System complexity
  – System diversity

• The basic assumptions of Unix/Windows/etc. are being re-evaluated.
Moore’s law: the free lunch

The power wall

AMD Athlon 1500 processor
http://www.phys.ncku.edu.tw/~htsu/humor/fry_egg.html
The power wall

• Power dissipation =
  Capacitive load × Voltage² × Frequency
  – Increase in clock frequency
    ⇒ mean more power dissipated
    ⇒ more cooling required
  – Decrease in voltage reduces dynamic power but increase the static power leakage

• We’ve reached the practical power limit for cooling commodity microprocessors
  – Can’t increase clock frequency without expensive cooling
The memory wall

1MHz CPU clock, 500ns to access memory in 1985

55% p.a.

10% p.a.
The ILP wall

- ILP = “Instruction level parallelism”
- Implicit parallelism between instructions in 1 thread
- Processor can re-order and pipeline instructions, split them into microinstructions, do aggressive branch prediction etc.
  - Requires hardware safeguards to prevent potential errors from out-of-order execution
- Increases execution unit complexity and associated power consumption
  - Diminishing returns
- Serial performance acceleration using ILP has stalled
End of the road for serial hardware

- Power wall + ILP wall + memory wall = brick wall
  - Power wall ⇒ can’t clock processors any faster
  - Memory wall ⇒ for many workloads performance dominated by memory access times
  - ILP wall ⇒ can’t keep functional units busy while waiting for memory accesses

- There is also a complexity wall, but chip designers don’t like to talk about it...
Multicore processors

• Multiple processor cores per chip
  – This is the future (and present) of computing
• Most multicore chips so far are shared memory multiprocessors (SMP)
  – Single physical address space shared by all processors
  – Communication between processors happens through shared variables in memory
  – Hardware typically provides cache coherence
Implications for software

The things that would have used this “lost” perf must now be written to use cores/accel

Historical 1-thread perf gains via improved clock rate and transistors used to extract ILP

#transistors still growing, but delivered as additional cores and accelerators
And the others?

Trends towards:

• **Heterogeneous processing**
  – Different types of core on a chip

• **System complexity**
  – Devices, interconnects, memory system

• **System diversity**
  – No two systems are alike
  – Very different performance tradeoffs

We’ll see this all very shortly...
Structure and Logistics
Assumptions

• C Programming
  – Practical labs are mostly in C.
  – You will be writing an OS.

• Computer Architecture
  – Need to know about hardware, registers, DMA,
  – You will be writing a device driver, pager, etc.

• Command line development
  – gcc, emacs/vi, make, etc.
  – Debugging facilities are primitive!
ETH courses
(or similar elsewhere)

• Parallel programming
  – Concurrency, locks, synchronization, ...
• Digital Design
  – Basics of hardware
• Computer Architecture and Systems Programming
  – C and assembly
  – Caches, devices, I/O, virtual memory, traps, ...
• Networks and Operating Systems
  – Memory management, protection, scheduling, device drivers...
• Systembau: alternative/complementary
  – Our emphasis: microkernels (practical) and advanced/research OS topics (theoretical)
Structure of the course

- Lectures
- Readings
- Project
Lectures

• Cover topics related to current project milestone.
  – Key ideas
  – History / influences

• Explanation / elaboration of readings

• Also Q&A if required
Readings

• Most of our material is not (or insufficiently) covered in textbooks
• Readings will be research papers
  – Seminal papers in the field
  – Background to the concepts you will deal with in the weeks’ project milestone
  – Context for the material covered in the lecture
Project

• **Build (most of) an OS**
  – based on Barrelfish research OS
  – PandaBoard hardware
Project Milestones

• Demonstrate new functionality
  – Important to keep up with the schedule!

• Milestone 0: this week
  – Familiarization, building, booting

• Each milestone covers one or more research concepts
  – E.g. self-paging, DSLs, multikernel architecture, URPC, ...

• Final codebase: complete OS
• Final milestone: documentation for the system
Evaluation: Project (65%)

- There are marks for each milestone
- Late milestones incur a penalty
- Beware milestones which build on previous ones
  - Don’t slip behind
Evaluation: Exam (35%)

• Date(s)
  – To be determined – sometime in February

• Material
  – Lectures, readings, and project material

• Format
  – Written examination (new this year!)
  – 90 minutes.

• We are reasonable people!
  – Test understanding and principles, not fiddly details
Dates and times

• Lecture:
  – Thursday 10:00 - 12:00 in CAB G.59

• Consultations and marking:
  – Friday 10:00 - 12:00 in CAB H.57

• Mailing list (for questions):
  – aos2014@lists.inf.ethz.ch
Logistics

• Initial milestone is done individually
• Project is undertaken in teams of 2
• Everyone gets a PandaBoard + cables each
  – Take care of them – they are fragile!
• So:
  – Please find a team partner before tomorrow’s session
  – You **MUST** attend tomorrow’s session to sign out your PandaBoard
  – If you can’t find a partner, you will be assigned one at the session.
People

• Timothy Roscoe

• Pravin Shinde

• Simon Gerber

• Kornilios Kourtis
What you need to do.

• Attend the session tomorrow!
  – Sign out your Pandaboard kits
  – Bring your student ID
  – Verify your board works before leaving

• Form groups of 2 for subsequent milestones
  – Inform Pravin by email before COB Friday

• Deliver your first (individual) milestone by next Friday
  – Come to the session and demo it
Hardware
Many thanks to ARM Ltd and Texas Instruments for their generosity in providing the hardware!
PandaBoard block diagram
The TI OMAP 4460 SoC
USB booting

“USB On-The-Go” port:
• Provides power from the host PC
• Host copies code over USB into PandaBoard memory
• Jumps to start of boot image
• No need for network, SD card, etc.

USB OTG port
Reset button
Serial console

- One byte at a time, bidirectional serial line
- Still the best low-level console interface for real OS hacking
- First milestone includes:
  - Very simple console output
  - Flash indicator LED

RS-232 interface

Indicator LEDs
ARM CPU cores

• 2 x ARM Cortex A9 cores
• SMP configuration
• We’ll start with just one core
• We’ll bring up the other one later

There are, in fact, two more ARM Cortex M3 cores. We’ll ignore these.

(unless you want to play with them in your spare time 😊)
ARM architecture overview

• Data Sizes:
  – 16 bits: Halfword
  – 32 bits: Word
  – 64 bits: Doubleword

• Note: different from Intel/AMD names!
## ARM Processor modes

Each mode has its own stack!

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervisor</td>
<td>Entered on reset or SWI instruction</td>
</tr>
<tr>
<td>FIQ</td>
<td>Fast interrupt</td>
</tr>
<tr>
<td>IRQ</td>
<td>Normal interrupt</td>
</tr>
<tr>
<td>Abort</td>
<td>Memory access violations</td>
</tr>
<tr>
<td>Undef</td>
<td>Undefined instructions</td>
</tr>
<tr>
<td>System</td>
<td>Privileged mode, same registers as user mode</td>
</tr>
<tr>
<td>User</td>
<td>User mode (regular processes)</td>
</tr>
</tbody>
</table>
ARM register sets  
(all 32 bits)
### ARM condition codes (some of them)

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Negative result</td>
</tr>
<tr>
<td>Z</td>
<td>Zero result</td>
</tr>
<tr>
<td>C</td>
<td>Carry from ALU</td>
</tr>
<tr>
<td>V</td>
<td>Overflow from ALU</td>
</tr>
<tr>
<td>I</td>
<td>Disable IRQ</td>
</tr>
<tr>
<td>F</td>
<td>Disable FIQ</td>
</tr>
<tr>
<td>mode</td>
<td>Current mode (4 bits)</td>
</tr>
</tbody>
</table>

Found in the CPSR/SPSR registers
Exception handling

1. CPSR $\rightarrow$ SPSR$^\text{mode}$
2. Set CPSR bits
   - New mode
   - Disable interrupts
     (if appropriate)
3. Return addr $\rightarrow$ LR$^\text{mode}$
4. Vector addr $\rightarrow$ PC

<table>
<thead>
<tr>
<th>Vector Address</th>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Reset</td>
</tr>
<tr>
<td>0x04</td>
<td>Undefined instruction</td>
</tr>
<tr>
<td>0x08</td>
<td>Software interrupt</td>
</tr>
<tr>
<td>0x0C</td>
<td>Prefetch abort</td>
</tr>
<tr>
<td>0x10</td>
<td>Data abort</td>
</tr>
<tr>
<td>0x14</td>
<td>reserved</td>
</tr>
<tr>
<td>0x18</td>
<td>IRQ</td>
</tr>
<tr>
<td>0x1C</td>
<td>FIQ</td>
</tr>
</tbody>
</table>

Base address
- $= 0x00$ or $0xFFFF0000$
ARM instruction set

- Classic RISC load/store architecture
  - ALU operations are register-register
- All instructions 32 bits long
- Most instructions are conditional
- On-die barrel shifter
- Branches are PC-relative

- LDR r0,[r1]
- STRNEB r2,[r3,r4]
- B <label>
- SUB r0,r1,#5
- ADD r2,r3,r3,LSL#2
- ADDEQ r5,r5,r6
OS: Barrelfish
What is Barrelfish?

• Open source research operating system
  – Fairly influential in research community
• ETH Zurich + Microsoft Research
• Currently supports:
  – 32- and 64-bit x86 AMD/Intel
  – Intel MIC/Xeon Phi (Knight’s Ferry)
  – ARMv5, Xscale, ARMv7m, ARMv7a
What things run on it?

- Many microbenchmarks
- Parallel benchmarks: Parsec, SPLASH-2, NAS
- Webserver: \url{http://www.barrelfish.org/}
- Databases: SQLite, PostgreSQL
- OpenSSH, other utils
- Virtual machine monitor
  - Linux kernel binary
- Microsoft Office 2010!
  - via Drawbridge
Why use Barrelfish?

• Very different to Linux, BSD, MacOS, Windows
  – The shock of the new
• Smaller OS
  – Easier to understand, easier to hack
• Multikernel approach (see later)
  – Simplifies multicore issues
• Includes lots of other people’s research ideas
  – Good illustration of any concepts
Some of the (non-original) ideas in Barrelfish

- Capabilities for resource management (KeyKOS, seL4)
- Minimize shared state (Tornado, K42)
- Upcall processor dispatch (Psyche, Sched. Activations)
- Push policy into user space (Exokernel, Nemesis)
- User-space RPC decoupled from IPIs (URPC)
- Lots of information (Infokernel)
- Single-threaded non-preemptive kernel per core (K42)
- Run drivers in their own domains (µkernels, Xen)
- Fast interprocess communication (LRPC, L4)
- Specify device registers in a little language (Devil)

We’ll see many of these later in the course
Toolchain
Toolchain

• C compiler: gcc
• Haskell compiler: ghc
• make
• Terminal emulator: picocom
• Boot utility: usbboot

• We provide a VirtualBox image with all this
• You are welcome to use your own platform
Building Barrelfish

$ mkdir ~/build_milestone_0
$ cd ~/build_milestone_0
$ /hake/hake.sh -s ~/barrelfish -a armv7

• Builds a Makefile for Barrelfish in the build_milestone_0 directory, based on the source tree in ~/barrelfish, for just the ARMv7 architecture.
• You can look at ./Makefile to see the result (but it can be very large...)

Systems@ETH Zürich
Building Barrelfish

$ mkdir ~/build_milestone_0
$ cd ~/build_milestone_0
$ /hake/hake.sh -s ~/barrelfish -a armv7
$ make aos_image
$ ls -l aos_image

- Make builds its own directories, dependencies, and other tools as well.
- Result ends up in the top of the build directory.
What’s this Hake thing?
(for the curious)

Scattered through the source tree are files called “Hakefile”

- Hake concatenates all of these and interprets them as a single, large Haskell expression
- This expression evaluates to the contents of `.\Makefile`

Why?

- Allows us to build multiple architectures from one source tree at the same time
- Power of a full programming language in expressing configurations
- Still get the efficiency of make for building
And finally, some advice
Important advice

• You will be graded on the design of your code
  – Does it work?
  – Handle corner cases, errors, invalid inputs, etc?
  – An operating system runs for a long time. Do you leak memory? Etc.
  – Have you thought about issues not explicit in the milestones, but important to a real OS?

• Not all these criteria can be well-specified in advance
  – Use common-sense in system design
  – You will be graded on issues you can think of and deal with
Important advice

You will be graded on the **quality** of your report

- Doxygen and other tools document *lines of code*, not the *design of a system*!
- Don’t submit generated documentation in place of a report.
- Describe the *choices* you made (and didn’t make)
- Talk about the tradeoffs
- Mention the *difficulties* and *challenges*, and how you overcame them.
- Show you understand how to build a system.
Important advice

• You will be examined on the depth of your *understanding*
  – Design principles
  – Why do techniques work?
  – When do they work (or not)?
  – What are the tradeoffs in a problem?
  – What factors affect the tradeoffs?
  – How might this change? How has it changed?
Final advice

• This course is a lot of fun, but a lot of work
  – and we are aware of this!
• It is important not to fall behind
  – If your team is struggling, ask for help.
• If you're good, it's tempting to be clever and cool
  – Resist this temptation!
  – Get the required work done before freestyling.
• We are here to help you!

Good luck and have fun!