Writing an operating system in 2.5 years

Yunhao Zhang
But first, writing an OS in one semester

- P0: understand C and user-level instructions
- P1: understand context-switch and multi-threading
- P2: understand exception and privilege levels
- P3: understand the disk abstraction
- P4: understand the file abstraction
- P5 (optional): understand I/O bus and devices
Why 2.5 years? An overview

- Motivation: Jun 2020 - Sep 2020
- Obstacles: Nov 2021 - Jan 2022
- Ideas: Jun 2022
- Implementation: Dec 2022
- Evaluation:
By June 2020, we only had egos-classic

~20K lines of code
Intel / Arm CPU
Linux / MacOS user process
20K lines of code
Students read a *very small* portion

2K lines of code
Students read a *large* portion
Intel x86 (1987)

CPU document has several thousands of pages

RISC-V (2010)

CPU document has <100 of pages
User-mode OS
Easier to compile and run

OS on real hardware
More realistic to play with
Motivations

~20K → 2K

x86 / ARM → RISC-V

Linux / MacOS → QEMU / board
Lesson

Good motivations should convince non-experts why the work is valuable.
Hello World

Summer 2020

Sep 12, 2020
ideal $\neq$ possible; OS $\neq$ hello-world

Obstacles

Summer 2020

Fall 2020
Obstacles & Hope

- Only 24KB memory
- No disk device
- Timer interrupt is supported
- Privilege levels and exceptions are supported
- CPU is well-documented and board is not too expensive
Obstacles & Hope: What to do?

- Need to modify the hardware design
- Need to write a kernel with the CPU support and documents
Background: Open-source hardware
Running open-source hardware

A binary file encoding the hardware design (clocks, registers, circuits, etc.)

FPGA emulates the hardware design
Idea #1: Increase the memory size

dcache = Some(DCacheParams(
  rowBits = site(SystemBusKey).beatBits,
  nSets = 256, // 16Kb scratchpad
  nWays = 1,
  nTLBEntries = 4,
  nMSHRs = 0,
  blockBytes = site(CacheBlockBytes),
  scratch = Some(0x800000000L)),

https://github.com/chipsalliance/rocket-chip/blob/b21c7879b3ea22f69cb8457109561f37c225f8ea/src/main/scala/subsystem/Configs.scala#L78
Background: SPI (simpler than USB)

### Table 64: SPI Instances

<table>
<thead>
<tr>
<th>Instance</th>
<th>Flash Controller</th>
<th>Address</th>
<th>cs_width</th>
<th>div_width</th>
</tr>
</thead>
<tbody>
<tr>
<td>QSPI 0</td>
<td>Y</td>
<td>0x10014000</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>SPI 1</td>
<td>N</td>
<td>0x10024000</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>SPI 2</td>
<td>N</td>
<td>0x10034000</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>
SPI: Serial Peripheral Interface

6 pins
GND + VCC + SPI (4)
Idea #2: Remap SPI1 to a microSD card

<table>
<thead>
<tr>
<th>Instance</th>
<th>Flash Controller</th>
<th>Address</th>
<th>cs_width</th>
<th>div_width</th>
</tr>
</thead>
<tbody>
<tr>
<td>QSPI 0</td>
<td>Y</td>
<td>0x10014000</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>SPI 1</td>
<td>N</td>
<td>0x10024000</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>SPI 2</td>
<td>N</td>
<td>0x10034000</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 64: SPI Instances
Idea #2: Remap SPI1 to a microSD card

## ChipKit SPI

```bash
class set_property -dict { PACKAGE_PIN G1 IOSTANDARD LVCMOS33 } [get_ports { ck_miso }]; 
#I0_L17N_T2_35 Sch=ck_miso
class set_property -dict { PACKAGE_PIN H1 IOSTANDARD LVCMOS33 } [get_ports { ck_mosi }]; 
#I0_L17P_T2_35 Sch=ck_mosi
class set_property -dict { PACKAGE_PIN F1 IOSTANDARD LVCMOS33 } [get_ports { ck_sck }]; 
#I0_L18P_T2_35 Sch=ck_sck
class set_property -dict { PACKAGE_PIN C1 IOSTANDARD LVCMOS33 } [get_ports { ck_ss }]; 
#I0_L16N_T2_35 Sch=ck_ss
```

## Pmod Header JA

```bash
class set_property -dict { PACKAGE_PIN G13 IOSTANDARD LVCMOS33 } [get_ports { ja_0 }]; 
#I0_0_15 Sch=ja[1]
class set_property -dict { PACKAGE_PIN B11 IOSTANDARD LVCMOS33 } [get_ports { ja_1 }]; 
#I0_L4P_T0_15 Sch=ja[2]
class set_property -dict { PACKAGE_PIN A11 IOSTANDARD LVCMOS33 } [get_ports { ja_2 }]; 
#I0_L4N_T0_15 Sch=ja[3]
class set_property -dict { PACKAGE_PIN D12 IOSTANDARD LVCMOS33 } [get_ports { ja_3 }]; 
#I0_L6P_T0_15 Sch=ja[4]
```

Find and replace these 4 wires in the hardware design.

Coming up with ideas is difficult

No progress at all for more than a year.
Not sure whether it can work eventually.
Being the only person pushing this work.
Lesson

Ideas are **difficult** to come up with and there is **no guarantee of success**.

Motivation | Obstacles | Ideas | Implementation | Evaluation
A bug taking >1 day to fix

core = RocketCoreParams(
    useVM = false,
    fpu = None,
    mulDiv = Some(MulDivParams(mulUnroll = 8)),
    btb = None,
    dcache = Some(DCacheParams(
        rowBits = site(SystemBusKey).beatBits,
        nSets = 256, // 16Kb scratchpad
        nWays = 1,
        nTLBEntries = 4,
        nMSHRs = 0,
        blockBytes = site(CacheBlockBytes),
        scratch = Some(0x8000000L)),
    icache = Some(ICacheParams(
        rowBits = site(SystemBusKey).beatBits,
        nSets = 64,
        nWays = 1,
        nTLBEntries = 4,
        blockBytes = site(CacheBlockBytes))))

https://github.com/chipsalliance/rocket-chip/blob/b21c7879b3ea22f69cb8457109561f37c225f8ea/src/main/scala/subsystem/Configs.scala#L78
Lesson
Implementing a system is *non-trivial*. It requires **hard work** and **determination**.
Lessons about doing research

• Good motivations should convince non-experts why the work is valuable.
• Ideas are difficult to come up with and there is no guarantee of success.
• Implementing a system is non-trivial, taking hard work and determination.
The full 4.5-year research process

2 years: Becoming familiar with OS education

Then, challenge the state-of-the-art

<table>
<thead>
<tr>
<th>Motivation</th>
<th>Obstacles</th>
<th>Ideas</th>
<th>Implementation</th>
<th>Evaluation</th>
</tr>
</thead>
</table>
Yunhaos Egos-2000 Packs an Entire RISC-V Operating System Into Just 2,000 Lines of Code

Designed to make it possible for students to learn about every aspect of OS development, egos-2000 is a miniature marvel.

Follow-up from OS hobbyists

Sipeed's Lichee RV64 board

https://github.com/cheofusi/egos-2000-d1
Future work

Enable **multi-core** in QEMU and implement **locks** in egos-2000

Leverage the **256MB** DDR memory and the **Ethernet** port on the Arty board
This project's vision is to help *every* college student read *all* the code of an operating system.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>153</td>
<td>boot loader &amp; tty driver</td>
<td>336</td>
<td>file system</td>
</tr>
<tr>
<td>240</td>
<td>sd card driver &amp; paging</td>
<td>320</td>
<td>applications &amp; system servers</td>
</tr>
<tr>
<td>32</td>
<td>interrupt &amp; exception handling</td>
<td>272</td>
<td>library &amp; networking (TBA)</td>
</tr>
<tr>
<td>108</td>
<td>page table &amp; software translation</td>
<td>64</td>
<td>makefile</td>
</tr>
<tr>
<td>341</td>
<td>timer, scheduler &amp; system call</td>
<td>134</td>
<td>RISC-V board &amp; emulator tools</td>
</tr>
</tbody>
</table>