Writing an operating system in 2.5 years

Yunhao Zhang
But first, writing an OS in 0.5 years

- P0: understand computer architecture
- P1: understand context-switch and multi-threading
- P2: understand interrupt and exception
- P3: understand privilege levels and protection
- P4: understand bus and I/O devices
- P5: understand file systems
Why 2.5 years? An overview

Motivation: Summer 2020
Obstacles: Fall 2020
Ideas: Fall 2021
Implementation: Spring 2022
Evaluation: Fall 2022
In June 2020, we have

\(~20K\) lines of code
run on Intel CPU
run on Linux / MacOS
20K lines of code
Students read a very small portion

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

CPU documents have **several thousands** of pages

RISC-V (2010)

CPU documents have **<100** of pages
User-mode OS

Easier to deploy and run

OS on hardware

More realistic and fun
3 motivations

- ~20K → ~2K
- Intel → RISC-V
- Linux / MacOS → real hardware
Lesson

Good motivations should convince non-experts why the work is important.

non-experts like students, my friend doing ML theory, etc.
Yet, ideal $\neq$ possible

Obstacles

Summer 2020  Fall 2020
Hello World

Obstacles

Summer 2020  ➔  Fall 2020

Sep 12, 2020
Obstacles & Hope

- The hardware had only 24KB memory
- There was no disk
- The hardware supports timer interrupt
- The hardware supports privilege levels
Overcome obstacles with ideas!

Idea #1
Increase memory

Idea #2
SD card extension

Obstacles

Fall 2020

Jan 2021

Nov 2021

Fall 2021

Ideas
Open-source hardware
Running open-source hardware

A binary file encoding the hardware design

Lithography:
a physical / chemical process
Idea #1: Increase memory

```scala
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>86</td>
<td><code>dcache = Some(DCacheParams(</code></td>
</tr>
<tr>
<td>87</td>
<td><code>rowBits = site(SystemBusKey).beatBits,</code></td>
</tr>
<tr>
<td>88</td>
<td><code>nSets = 256, // 16Kb scratchpad</code></td>
</tr>
<tr>
<td>89</td>
<td><code>nWays = 1,</code></td>
</tr>
<tr>
<td>90</td>
<td><code>nTLBEntries = 4,</code></td>
</tr>
<tr>
<td>91</td>
<td><code>nMSHRs = 0,</code></td>
</tr>
<tr>
<td>92</td>
<td><code>blockBytes = site(CacheBlockBytes),</code></td>
</tr>
<tr>
<td>93</td>
<td><code>scratch = Some(0x80000000L)),</code></td>
</tr>
</tbody>
</table>
```

https://github.com/chipsalliance/rocket-chip/blob/b21c7879b3ea22f69cb8457109561f37c225f8ea/src/main/scala/subsystem/Configs.scala#L78
Idea #2: Background

<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

Chapter 19 of Sifive FE310 manual, v19p04
Remap SPI1 to Pmod1 for 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
Remap SPI1 to Pmod1 for microSD card

Old SPI1 mapping

New SPI1 mapping

6 pins
GND + VCC + SPI (4)
Modifying the hardware design

## ChipKit SPI

```tcl
set_property -dict { PACKAGE_PIN G1  IOSTANDARD LVCMOS33 } [get_ports { ck_miso }];
set_property -dict { PACKAGE_PIN H1  IOSTANDARD LVCMOS33 } [get_ports { ck_mosi }];
set_property -dict { PACKAGE_PIN F1  IOSTANDARD LVCMOS33 } [get_ports { ck_sck }];  #
set_property -dict { PACKAGE_PIN C1  IOSTANDARD LVCMOS33 } [get_ports { ck_ss }];  #
```

## Pmod Header JA

```tcl
set_property -dict { PACKAGE_PIN G13  IOSTANDARD LVCMOS33 } [get_ports { ja_0 }];  #IO_A[0]
set_property -dict { PACKAGE_PIN B11  IOSTANDARD LVCMOS33 } [get_ports { ja_1 }];  #IO_B[1]
set_property -dict { PACKAGE_PIN A11  IOSTANDARD LVCMOS33 } [get_ports { ja_2 }];  #IO_C[2]
set_property -dict { PACKAGE_PIN D12  IOSTANDARD LVCMOS33 } [get_ports { ja_3 }];  #IO_D[3]
```

[Link to repository]

Find these 4 wires in the repo and replace them.
Road towards ideas is **difficult**

- **No concrete progress** for >1 year
- **Not sure** whether this can succeed at all
- **Only person** working on this project

- **Obstacles**
  - Fall 2020

- **Ideas * 2**
  - Fall 2021
Take-away:
Ideas are **difficult** to come up with and there is **no guarantee of success** 😞
Iteration #0
* [2020.09] Setup the toolchain provided by SiFive; Compile and run Hello World on Arty
* [2020.09] Test the basic input and print functionalities using the SiFive Metal library

Iteration #1
* [2020.12] Increase the processor memory from 24KB to 160KB (128KB + 32KB)
* [2020.12] Confirm that the memory cannot be further increased due to the limitation of Artix-7 35T FPGA chip

Iteration #2
* Yeah, I didn’t work on this project in most of 2021...
* [2021.12] Create a docker image for portable toolchain setup: [Docker Hub repo](https://hub.docker.com/repository/docker/yhzhang0128/arty-toolchain)
* [2021.12] Reconnect the processor SPI bus controller to the Arty Pmod1 pins
* [2021.12] Implement the SD card initialize, read and write functions
* [2021.12] Increase the processor clock frequency from 32MHz to 65MHz so that read/write blocks become faster
Iteration #3
* [2022.01] Confirm that the processor clock frequency cannot be further increased
* [2022.01] Implement the `dev_tty`, `dev_disk`, `cpu_intr` and `cpu_mmu` interfaces in earth
* [2022.01] Load the ELF format grass kernel binary file from the SD card to memory
* [2022.01] Implement the control transfer from earth to grass kernel and then to a user application
* [2022.01] Implement `mkrom` so that creating the bootROM image no longer require Vivado or Docker

Iteration #4
* [2022.02] Read Chapter 1, 2 and 3 of [RISC-V manual](riscv-privileged-v1.10.pdf)
* [2022.02] Read Chapter 8, 9 and 10 of [FE310 manual](si-five-fe310-v19p04.pdf)
* [2022.02] Implement timer reset, preemptive scheduling and inter-process communication
* [2022.02] Implement the kernel processes: GPID_PROCESS, GPID_FILE, GPID_DIR, GPID_SHELL
Iteration #5
* [2022.03] Implement system calls and 4 shell commands: `pwd`, `ls`, `cat` and `echo`
* [2022.03] Add support of background shell commands and `killall`
* [2022.03] Add servers in `library` and cleanup access to the file system
* [2022.03] Cleanup the code controlling UART and SPI; Remove dependency on the Metal library

Iteration #6
* [2022.04] Add a simple boot loader `earth/earth.S`; Remove dependency on the Metal library
* [2022.04] Add `_write()` and `_sbrk()` in `library/libc`; Remove the Metal library entirely
* [2022.04] Enrich `struct grass` in order to improve clarity of the architecture
* [2022.04] Experiment with Physical Memory Protection (PMP) and switching privilege level (machine <-> user)
The bug taking me >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 and requires determination and hard work
Evaluating the ideas: a class!

Remember to write the course evaluations!

What can be improved?
A 4.5-year research process

2 years: Becoming familiar with OS education

Then, challenge state-of-the-art

Motivation  Obstacles  Ideas  Implementation  Evaluation

Summer 2018  Summer 2020

Summer 2020  Fall 2022
Publish the research

Fighting for a world where every college student can read all the code of an operating system

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>199</td>
<td>Boot Loader &amp; TTY Driver</td>
<td>336</td>
<td>File System</td>
</tr>
<tr>
<td>182</td>
<td>SD Card Driver</td>
<td>264</td>
<td>Applications &amp; Daemons</td>
</tr>
<tr>
<td>32</td>
<td>Interrupt &amp; Exception Handling</td>
<td>269</td>
<td>Library &amp; Networking (TBA)</td>
</tr>
<tr>
<td>137</td>
<td>Page Table &amp; Software Translation</td>
<td>64</td>
<td>Makefile</td>
</tr>
<tr>
<td>345</td>
<td>Timer, Scheduler &amp; System Call</td>
<td>172</td>
<td>RISC–V Emulator &amp; Board Tools</td>
</tr>
</tbody>
</table>
Stay in touch: LinkedIn or Instagram
Homework

• P4 is optional

• P5 is due on Dec. 7 (extended)

• Please help with course evaluations!
  • see the pinned post #276 on Ed discussion