Lab4: Preemptive Multitasking

Mansour Alharthi
Overview

• Lab 4 is LONG.
Overview

- Lab 4 is **LONG**. 3325 lines, 19756 words, and 134,143 chars!
- Due: < 5 weeks from now (Apr 13)
Our OS (Up to Lab3)
Our OS (Up to Lab3)

Almost everything is here 😞
Our OS (Up to Lab3)

Wait, when did we move to EL2 from EL3?
Our OS (Up to Lab3)

```
switch_to_el2:
    // switch to EL2 if we're in EL3. otherwise switch to EL1
    cmp    x0, 0b11   // EL3
    bne    switch_to_el1

    // set-up SCR_EL3 (bits 0, 4, 5, 7, 8, 10) (A53: 4.3.42)
    mov    x2, #0x5b1
    msr    SCR_EL3, x2

    // set-up SPSR and PL switch! (bits 0, 3, 6, 7, 8, 9) (ref: C5.2.20)
    mov    x2, #0x3c9
    msr    SPSR_EL3, x2
    adr    x2, switch_to_el1
    msr    ELR_EL3, x2
    eret
```

kern/src/init/init.s
Our OS (Up to Lab3)

EL0

EL1

EL2

EL3

User

Kernel

Hypervisor

Monitor

FAT

Memory Allocator

Shell
Our OS (after Lab4)

EL0: P P P P P P P

EL1: FAT Memory Allocator Shell Exception Handler IRQ Virtual Memory Management Process Scheduler

EL2

EL3

User

Kernel

Hypervisor

Monitor
Overview

**Phase 1:** ARM Architecture (5 subphases).

**Phase 2:** It’s a Process (5 subphases).

**Phase 3:** Memory Management Unit (2 subphases).

**Phase 4:** Programs In The Disk (2 subphases).
Our OS – Before Lab 4

EL0

EL1

EL2

EL3

Hardware

Timer

SD Card

DRAM

FAT
Memory Allocator
Shell
Our OS – Lab4 Phase 1

1. Move to EL1
2. Configure ARM exception vector.
1. Create incomplete process implementation

2. Configure IRQ with timer interrupts.

3. Create a process scheduler & process queue.

4. Implement sleep syscall handler.

- **EL0**: P, P
- **EL1**: FAT, Memory Allocator, Shell, Exception Handler, IRQ, Process Scheduler
- **EL2**: 
- **EL3**: 

Hardware:
- Timer
- SD Card
- DRAM
Our OS – Lab4 Phase 3

1. Switch processes to use virtual addresses.

2. Create VMM & enable MMU.

3. Create MMU Page tables as per our design.
Our OS – Lab4 Phase 4

1. Complete process implementation by loading programs from disk.

2. Implement more syscall handlers & run user level programs from the disk.
Our OS – After Lab4

- EL0
  - P
  - P
  - P
  - P
  - P

- EL1
  - FAT
  - Memory Allocator
  - Shell
  - Exception Handler
  - IRQ
  - Process Scheduler
  - Virtual Memory Management

- EL2

- EL3

Hardware

- Timer
- MMU
- SD Card
- DRAM
Flow of our process scheduler (context switching)

Initialize:

- IRQ
- Timer
- Process Scheduler
- Exception Handler
Flow of our process scheduler (context switching)

Inside the scheduler:
1. Save P1 trap frame
2. Push P1 to end of Q
3. Find the next Ready (P2)
4. Set trap frame to P2

Call
Return
Virtual Vs Physical Layout (Kernel View)

- **Virtual Space**:
  - 0x0 - 0x4000_0000
  - IO
  - All regions mapped during kernel initialization
  - Allocated on demand using Allocator

- **Physical Space**: 0x0 - 0x4000_0000
  - IO
  - memory_map() end

```c
memory_map()
```

---

**Notes**

- The virtual space is mapped to the physical space during kernel initialization.
- Regions can be allocated on demand using Allocator.
Virtual Vs Physical Layout (User View)

Virtual Space

Physical Space

On demand page mapping and allocation using allocator

memory_map() end

0x0000_0000

0x3F00_0000

0x4000_0000

0xffff_ffff_ffff_ffff

0xffff_ffff_c000_0000

0xffff_ffff_ffff_ffff

0x0000_0000
Q&A