Week 6
CS4973/CS6640
02/10 2025
https://naizhengtan.github.io/25spring/

- [x] 1. last time: egos
- [x] 2. egos exception handling
- [x] 3. syscall in egos
- [ ] 4. CPU privilege levels
- [ ] 5. System virtualization

- Lab4: OS protection

RISC-V

earth: M
grass + sys procs: S
other apps: U

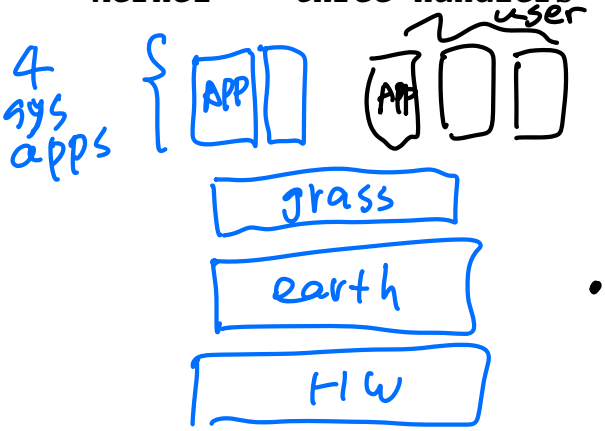
most → least
diff:
① instructions
② registers
③ memory

1. last time: egos

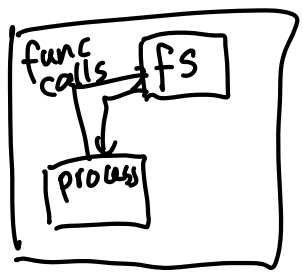
- [x] OS organization
- [x] egos design
- [x] egos boot

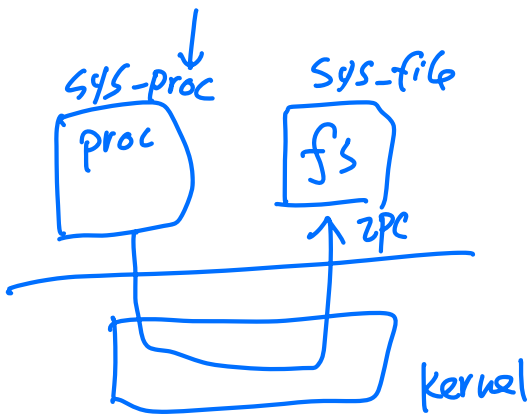
* kernel ~= three handlers

4 sys apps



monolithic-kernel, micro-kernel





- boot:

CPU
↳ 0x2040_000
↳ earth.S
↳ main (earth.c)
↳ main (grass.c)
↳ sys_proc
↳ sys_file
↳ sys_dir
↳ sys_shell
↳ $ ← 🧍

Q: 3 ways to trap to kernel.?

- interrupt
- syscall
- exception

(a) Interrupt

what code

CSR

Q: How does the CPU determine what to execute when an interrupt is triggered?

mtvec

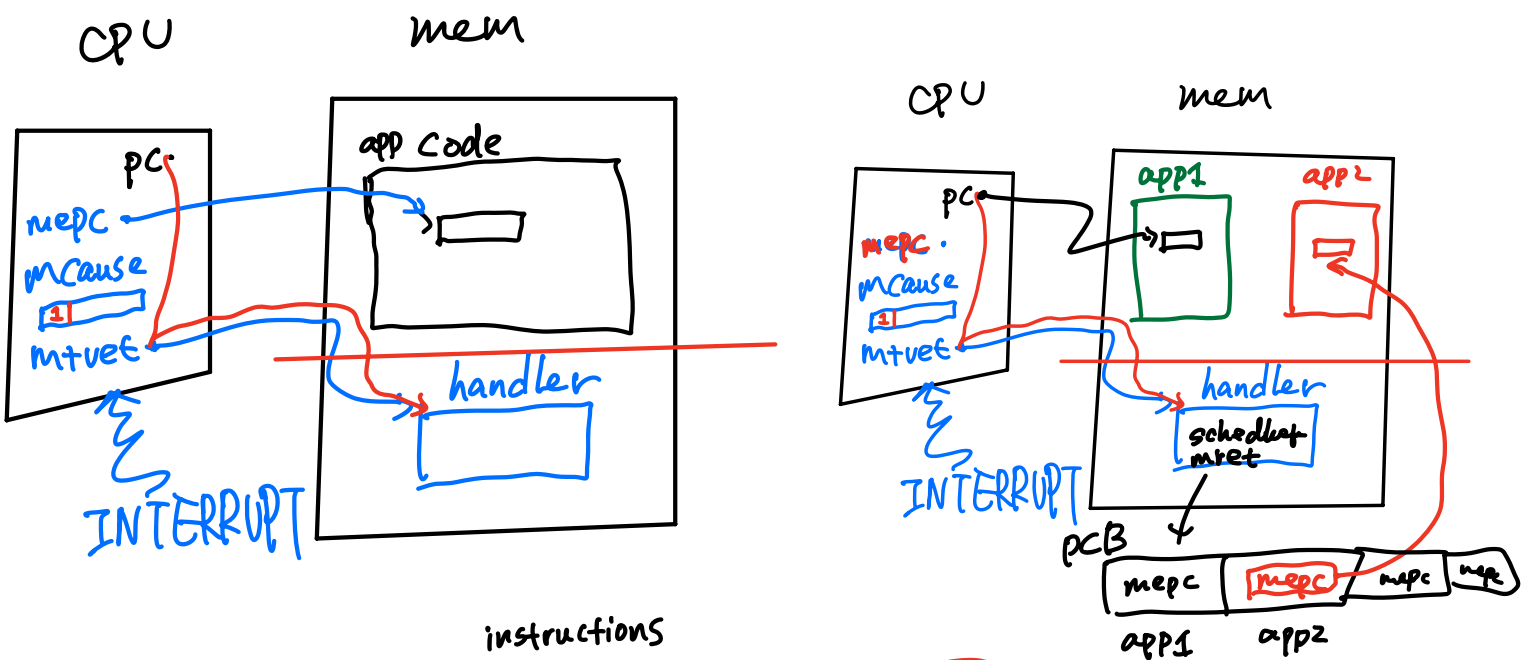
Q: After handling an interrupt, where does the CPU resume execution?

mepc

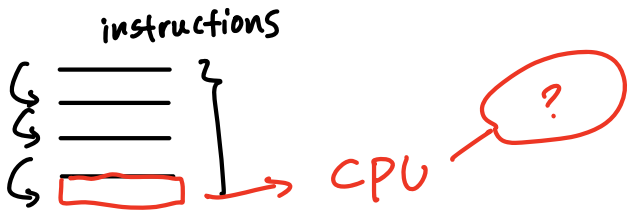
mret

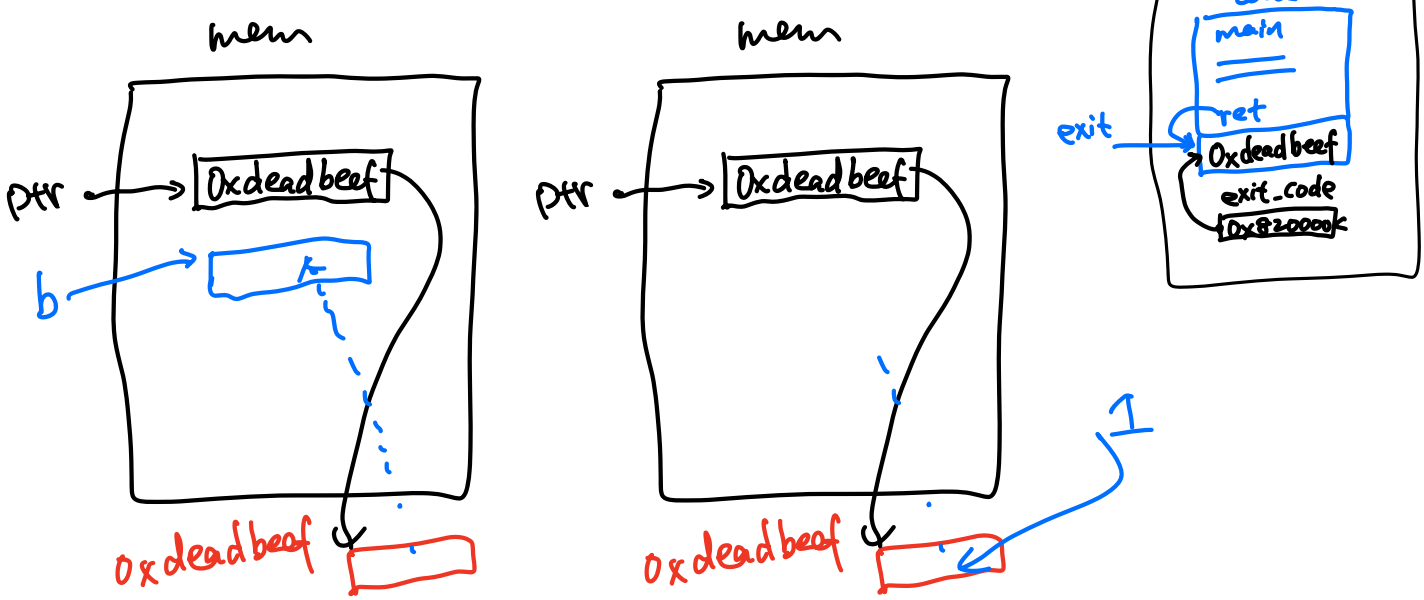
Q: How does the kernel identify which interrupt was triggered?

mcause



(b) Exception





Interrupt	Exception Code	Description
1	0	*Reserved*
1	1	Supervisor software interrupt
1	2	*Reserved*
1	3	Machine software interrupt
1	4	*Reserved*
1	5	Supervisor timer interrupt
1	6	*Reserved*
1	7	Machine timer interrupt
1	8	*Reserved*
1	9	Supervisor external interrupt
1	10	*Reserved*
1	11	Machine external interrupt
1	12–15	*Reserved*
1	≥16	*Designated for platform use*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	*Reserved*
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved*
0	15	Store/AMO page fault
0	16–23	*Reserved*
0	24–31	*Designated for custom use*
0	32–47	*Reserved*
0	48–63	*Designated for custom use*
0	≥64	*Reserved*

NOW → syscall

timer →

"ecall"

lab4 →

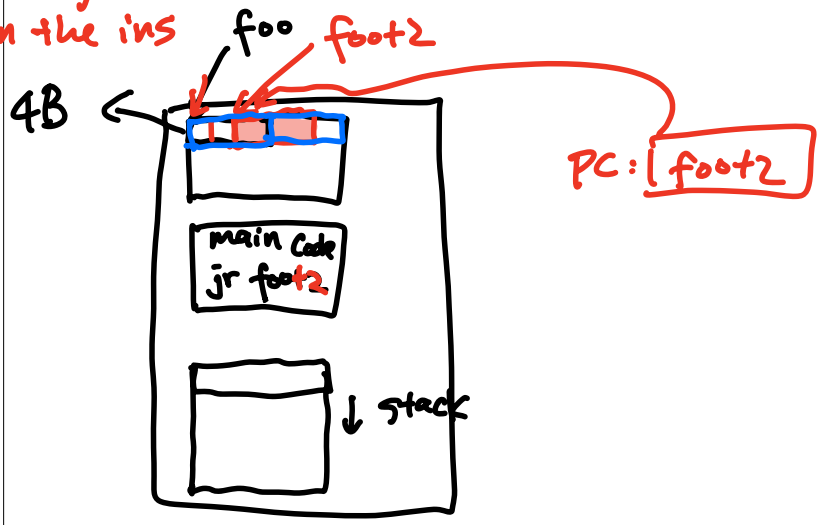
lab5 →

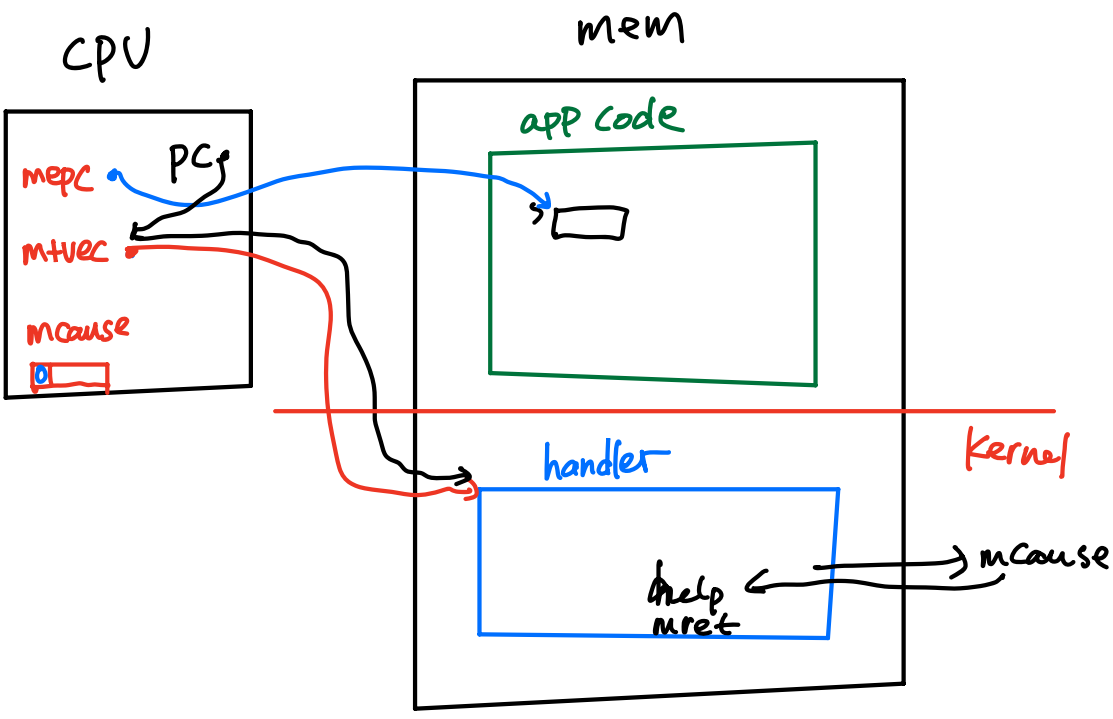
① not well-formed instruction

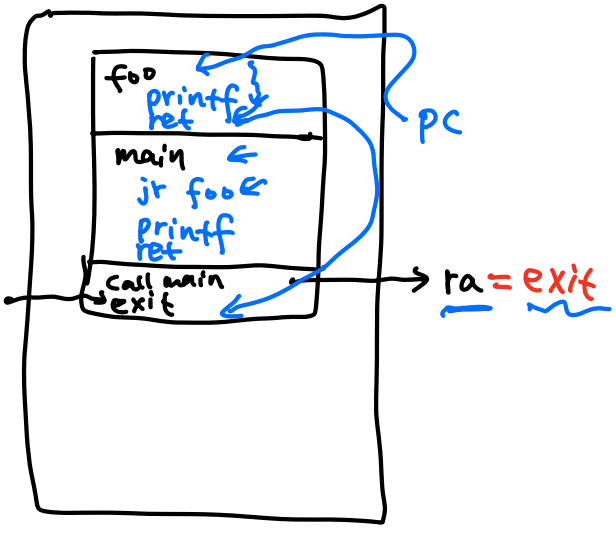
② no privileged to run the ins

egos-2k+ syscall workflow

```
sys_send/recv (syscall.c)
  |                                                          ^
  +-> sys_invoke (syscall.c)                               [ret]
        |                              USER SPACE            |
-------[trap]-------------------------------------------------|----
        |                              KERNEL                |
        +-> trap_entry (cpu_intr.c)                        [mret]
            |                                                |
            +-> trap_handler (kernel.c)                    [ret]
                [switch to kernel stack]       [switch to user stack]
                ...                                        |
                |                                          |
                +-> proc_syscall (syscall.c)               [ret]
                    |                                      |
                    +-> proc_send/recv (syscall.c)         [ret]
                        |                                  |
                        +-> sys_yield (scheduler.c)        [ret]
                            |                              |
                            +->[context switch]      [switch back]
                                  |                        |
                                  +-->[other proc running]--+
```




CPU
mem
app code
mepc
PC
mtvec
mcause
0
kernel
handler
mcause
help
mret


foo
printf
ret
main
jr foo
printf
ret
call main
exit
pc
ra = exit

Linux:

(c) Syscall

fs →
- open/close/read/write
- mkdir/chmod/stat

- socket/bind/connect/listen
- pipe/dup2

Q1: how does control transfer to the kernel? (trap to kernel)

exception

interrupt

Q2: how does kernel understand what the application wants?
That is, what information is needed for handling a syscall?

pid (x)
mepc (x)

~~addr~~ • ~~type~~ id
• arguments

Q3: where is syscall-related information stored?

• registers
OR
• mem { stack / else

egos:
send/recv/yield

egos syscall trap:
* 0x200_0000 = 1
↳ MSIP of $mip
↳ pending interrupt
↳ software int

Q1: sys_invoke()

Q2: struct syscall

Q3: well-known mem location

[skipped]

Q: How could I know what privilege level the current CPU is running in?

"""
RISC-V deliberately doesn't make it easy for code to discover what mode it is running it because this is a virtualisation hole. As a general principle, code should be designed for and implicitly know what mode it will run in. Applications code should assume it is in U mode. The operating system should assume it is in S mode (it might in fact be virtualised and running in U mode, with things U mode can’t do trapped and emulated by the hypervisor).
"""

[from https://forums.sifive.com/t/how-to-determine-the-current-execution-privilege-mode/2823]