Project #2: System Calls

Hyungjoon Kwon
(hishine6@snu.ac.kr)
Systems Software &
Architecture Lab.
Seoul National University

2024.03.25

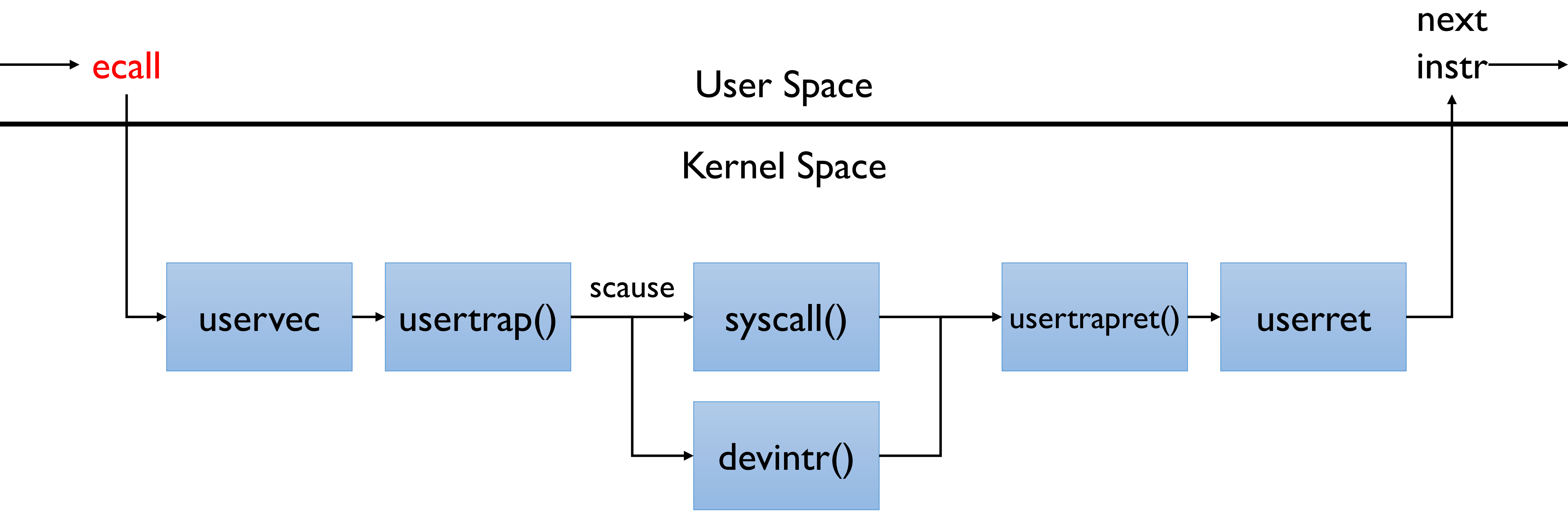
System Calls

- User applications can access the operating system kernel in a restricted way
- The interfaces that allow user applications to request services from the operating system kernel
- The operating system kernel does the requested task on behalf of user applications

Three RISC-V privilege modes

- Machine Mode
  - CPU starts in machine mode
- Supervisor Mode
  - Allowed to execute privileged instructions
    - Enable/Disable interrupts
    - Modify the page table base register
    - …
  - The operating system kernel runs in supervisor mode
- User Mode
  - User processes run in user mode

Traps from User Space (U-mode → S-mode)

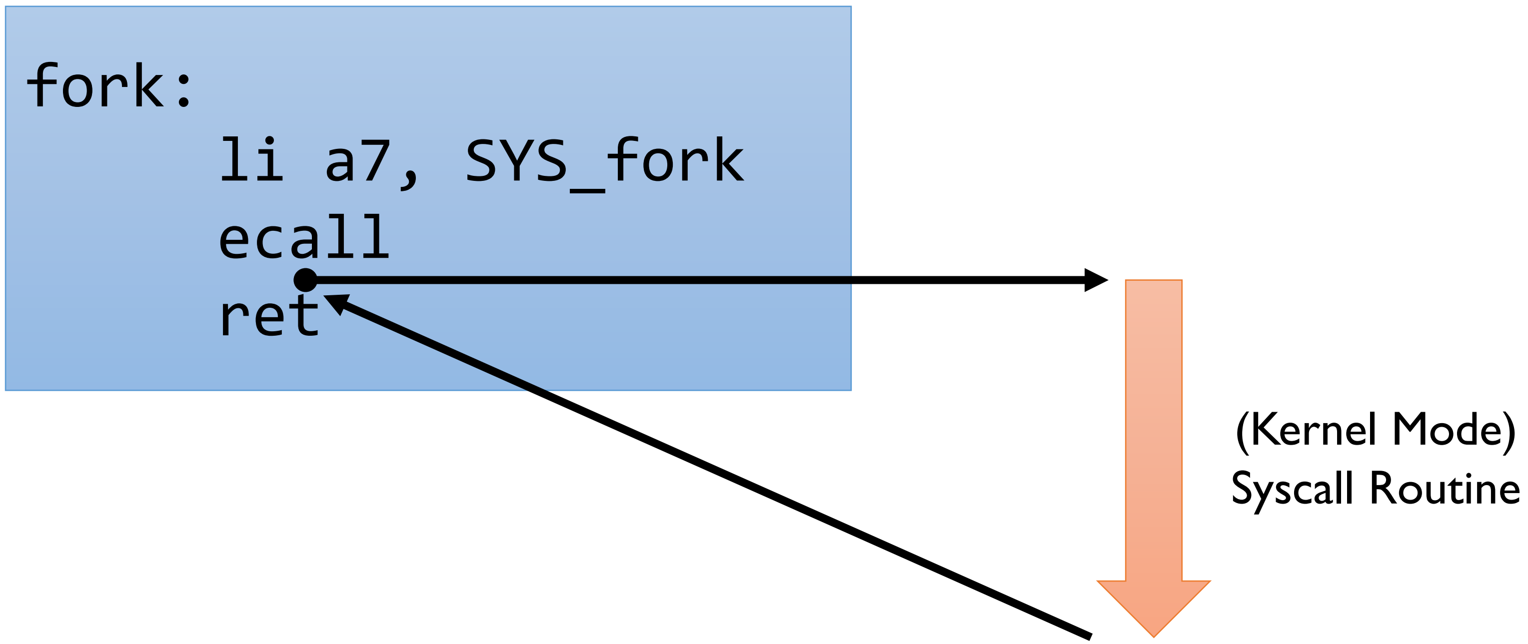


Some Registers

- satp
  - Pointer to page table
- scause (mcause)
  - Event which caused a trap
- sepc (mepc)
  - Program counter when a trap occurs
- sscratch (mscratch)
  - A dedicated register for use by supervisor (machine) mode
- stvec (mtvec)
  - Pointer to trap vector

ecall

- User applications execute the ecall instruction to invoke system calls
- E.g., fork()



What happens on ecall

- The RISC-V hart performs all these steps as a single operation
  - Copy the **`pc`** into **`sepc`**
  - Set **`scause`** to reflect the trap's cause
  - Set the **`stval`** if necessary (e.g., fault address)
  - Set the mode to supervisor
  - Copy **`stvec(which is uservec in xv6)`** to the **`pc`**
  - Start executing at the new **`pc`**
  - Note: the hart doesn't save any registers other than the **`pc`**

uservec

- Start in supervisor mode
- Save registers values to trapframe
  - Hart only saves the PC register
- Initialize kernel stack pointer
- Install the kernel page table
- Jump to usertrap()

usertrap()

- Install the kernel trap vector
- Save user program counter
- Handle an interrupt, exception, or system call depending on the value of scause register
- Call usertrapret() when it is done

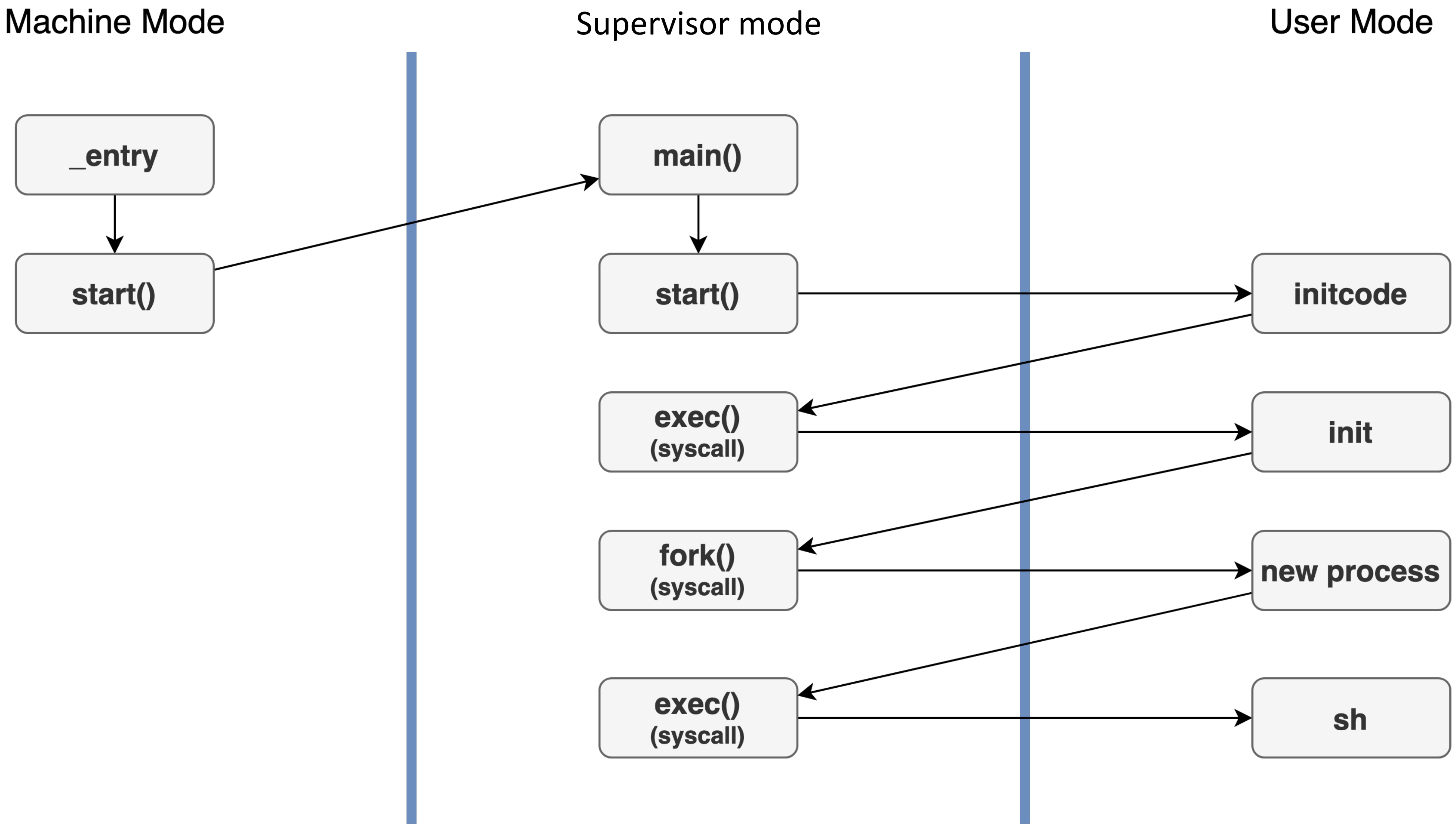
usertrapret()

- Install the user trap vector
- Restore user program counter
- Jump to userret

userret

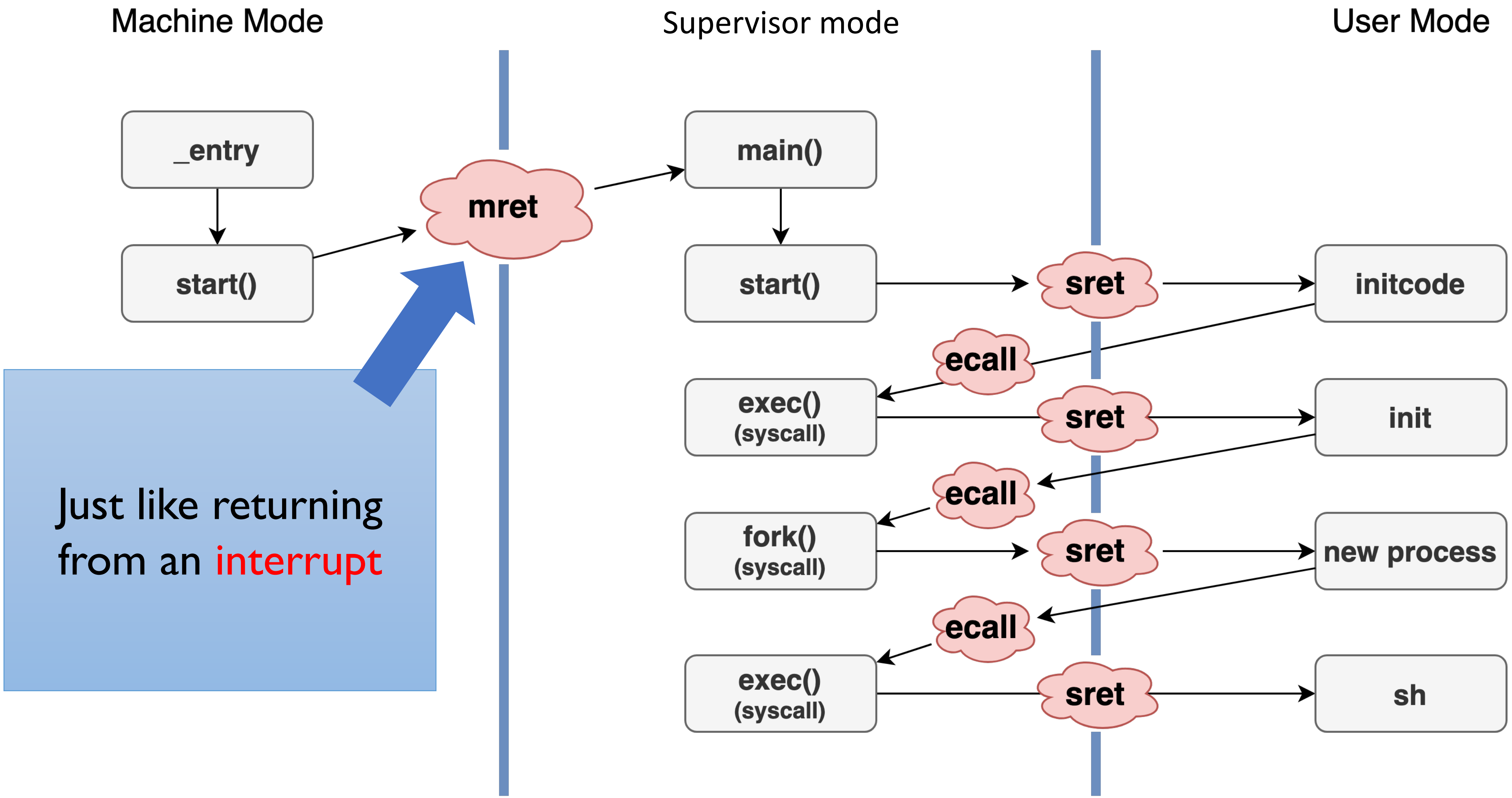
- Switch to the user page table
- Restore registers from trapframe
- Return to user mode (sret)

xv6 booting



Machine Mode
Supervisor mode
User Mode
_entry
start()
main()
start()
initcode
exec()
(syscall)
init
fork()
(syscall)
new process
exec()
(syscall)
sh

xv6 booting



Machine Mode
Supervisor mode
User Mode
_entry
start()
mret
main()
start()
sret
initcode
ecall
exec()
(syscall)
sret
init
ecall
fork()
(syscall)
sret
new process
ecall
exec()
(syscall)
sret
sh
Just like returning
from an interrupt

Trap delegation

- By default, all traps at any privilege level are handled in M-mode
- Register medeleg and mideleg can set certain traps to be processed directly by a lower privilege level (S-mode)
- Setting a bit in medeleg or mideleg will delegate the corresponding trap, when occurring in S-mode or U-mode, to the S-mode trap handler.

Project #2-1 (30 points)

- Your task is to implement a new system call named kbdints()
- It returns the total number of keyboard interrupts from the console input device since boot
- System call number assigned to 22 (already done in branch pa2)

Project #2-2 (70 points)

- Your task is to implement a new system call named time()
- It returns the value of the mtime register
- System call number assigned to 23 (already done in branch pa2)
- Assembly instruction to read the mtime register

```
# reading the mtime register
    ...
    rdtime  a0
    ...
```

- The rdtime instruction or reading the mtimer register is only available in the RISC-V machine mode

Project #2

- Tips
  - Read Chap. 4.1 of the xv6 book to understand RISC-V's privileged modes and trap handling mechanism
    (More detailed information can be found in the RISC-V Privileged Architecture manual)
  - Read Chap. 4.2 ~ 4.5 of the xv6 book to see how traps are handled in xv6
  - Read Chap. 5.1 ~ 5.4 of the xv6 book to learn about hardware interrupts

Project #2

- You may want to consult:
  - kernel/console.c
    - Console related function handling
  - kernel/syscall.{c, h}
    - General system call handling
  - kernel/sysproc.c
    - Several system call implementations
  - kernel/trap.c
    - Trap handling
  - kernel/kernelvec.S
    - M-mode, S-mode interrupt vectors
  - kernel/start.c
    - xv6 kernel boot up code
  - And other files if necessary

Project #2 (updated on 3/26)

- Restrictions
  - We found that the rdtime instruction is not supported or does not behave correctly in older versions of qemu
    - qemu version 8.2.0 or later ($ qemu-system-riscv64 –version)
  - We will run qemu-system-riscv64 with the –icount shift=0 option, which enables aligning the host and virtual clocks. This setting is already included in the Makefile for pa2 branch.
  - You can assume a uniprocessor RISC-V system (CPUS = 1) for this project assignment
  - You Should not modify the mcounteren register
  - For kbdints, count should be initialized to 0 on boot
  - Do not change the system call number for kbdints and time
  - You only need to change the files in the kernel directory

Project #2

- Skeleton Code
  - You should work on the pa2 branch of the xv6-riscv-snu repository as follows:

```
$ git clone https://github.com/snu-csl/xv6-riscv-snu
$ git checkout pa2
```

  - The pa2 branch has a user-level utility program named kbdints, time which can be built from the user/kdints.c, user/time.c file

Project #2

- Due
  - 11:59 PM, April 7 (Sunday)

- Submission
  - Run the make submit command to generate a tarball named xv6-pa2-{STUDENTID}.tar.gz in the xv6-riscv-snu directory
  - Upload the compressed file to the submission server
  - The total number of submissions for this project will be limited to 30
  - Only the version marked FINAL will be considered for the project score
  - In this project, you do not need to submit a report

Thank you!