Project #2: System Calls

Injae Kang
(abcinje@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

2023.09.26.

System Calls

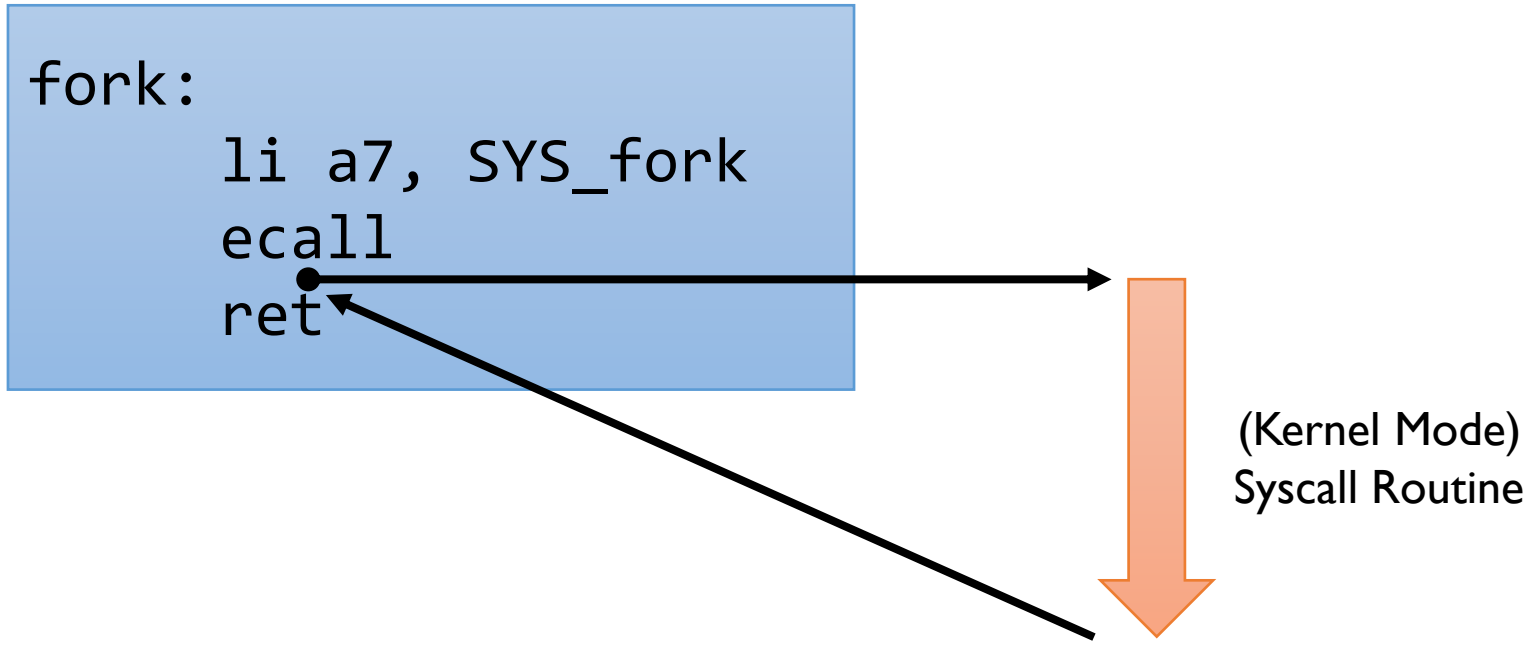
- User applications can access the operating system kernel in a restricted way
- The interfaces that allow user applications to request services from the operating system kernel
- The operating system kernel does the requested task on behalf of user applications

Three RISC-V privilege modes

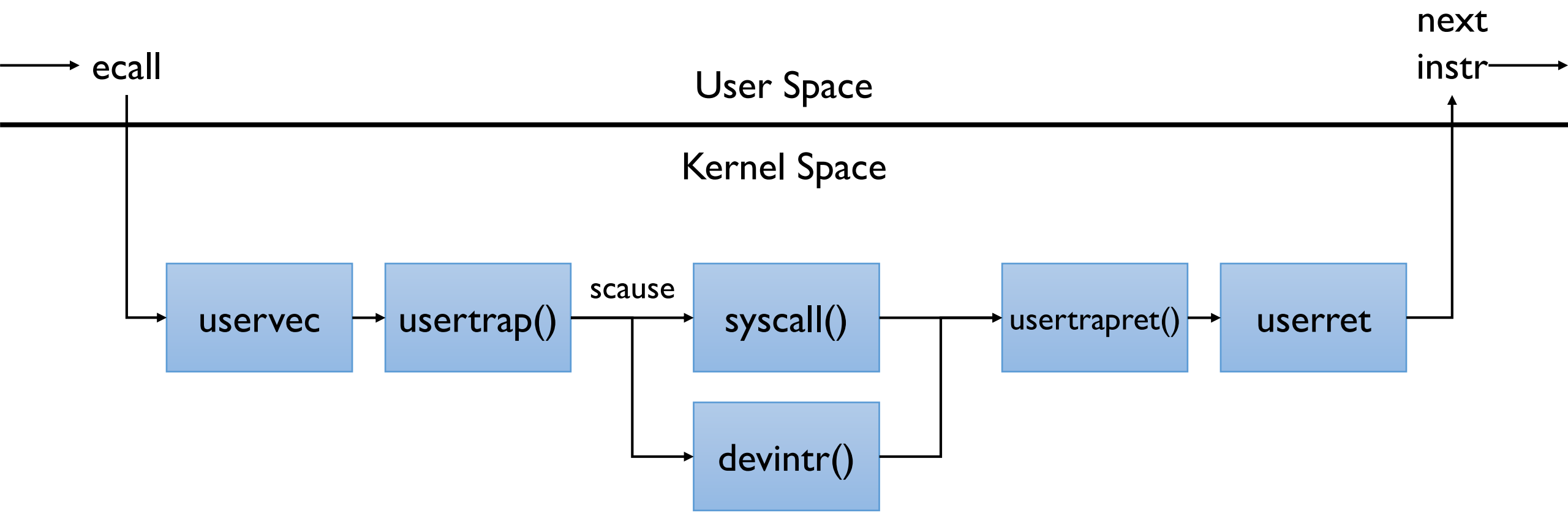
- Machine Mode
  - CPU starts in machine mode
- Supervisor Mode
  - Allowed to execute privileged instructions
    - Enable/Disable interrupts
    - Modify the page table base register
    - …
  - The operating system kernel runs in supervisor mode
- User Mode
  - User processes run in user mode

ecall

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



Traps from User Space



uservec

- Start in supervisor mode

- Save registers values to trapframe
- 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
- Set privilege mode to user
- Restore user program counter
- Jump to userret

userret

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

Some Registers

- satp
  - Pointer to page table
- scause
  - Event which caused a trap
- sepc
  - Program counter when a trap occurs
- sscratch
  - Backup of a0 register
- stvec
  - Pointer to trap vector

Project #2

- Your task is to implement a new system call named ntraps()
- It returns the number of system calls or interrupts invoked

- int ntraps(int type);
  - ntraps(0) returns the total number of system calls made for all cores (including the current ntraps() syscall)
  - ntraps(1) returns the total number of device interrupts received by all cores (including timer interrupts)
  - ntraps(2) returns the total number of timer interrupts received by all cores
  - For the other type values, it returns -1

Project #2

- You may want to consult:
  - kernel/defs.h
    - Function definitions
  - kernel/syscall.{c, h}
    - General system call handling
  - kernel/sysproc.c
    - Several system call implementations
  - kernel/trap.c
    - Trap handling
  - kernel/spinlock.{c, h}
    - Spinlock implementation
  - and other files if necessary

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

- Restrictions
  - Each count should be initialized to 0 on boot
  - Do not change the system call number for ntraps(), which is already assigned to 22
  - You only need to change the files in the kernel directory
  - Do not change the kernel/start.c file
  - The implementation must behave correctly even on multi-core systems

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 ntraps, which can be built from the user/ntraps.c file

Project #2

- Due
  - 11:59 PM, October 8 (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!