

Gahyun Lee

([windwaker@snu.ac.kr](mailto:windwaker@snu.ac.kr))

Systems Software &  
Architecture Lab.  
Seoul National University

2026.03.19.

# Project #2: System Calls



# 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
- Can access all hardware registers

## ■ Supervisor Mode

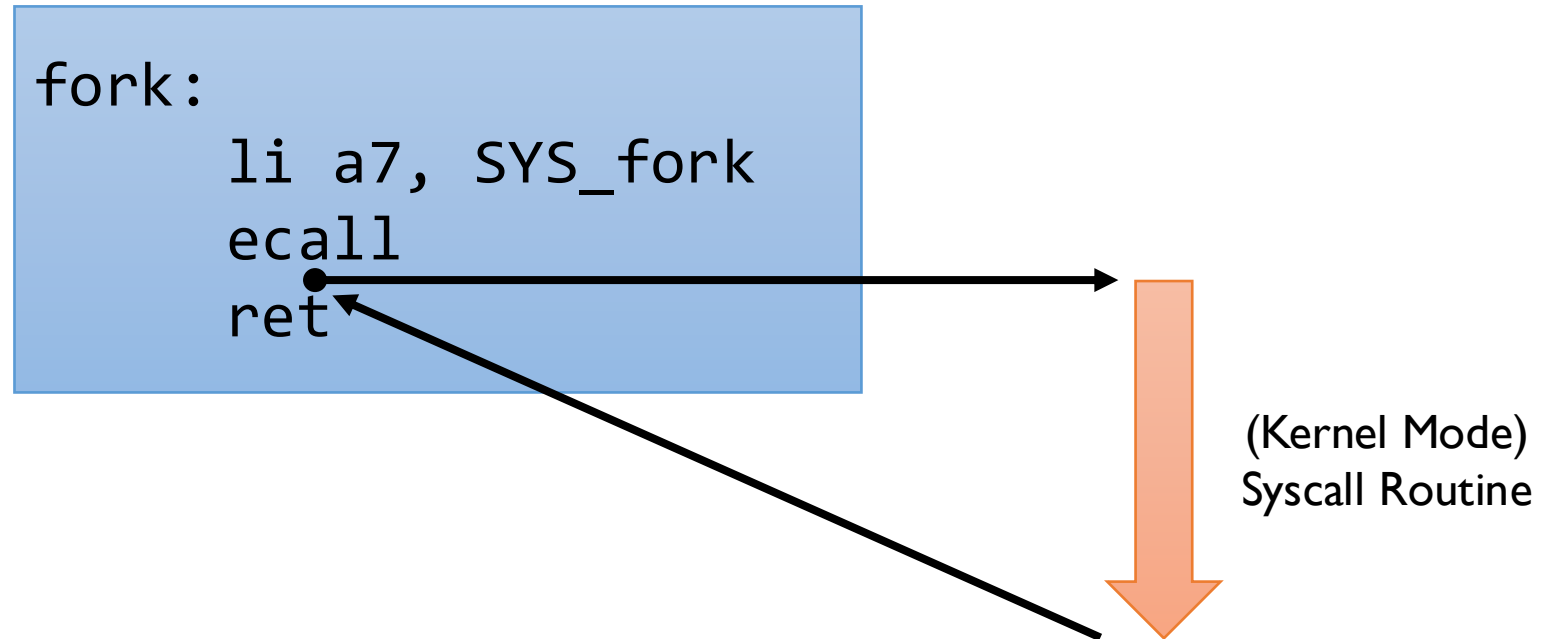
- Allowed to execute privileged instructions
  - Enable/Disable interrupts
  - Modify the page table base register
  - ...
- The operating system kernel runs in supervisor mode

## ■ User Mode

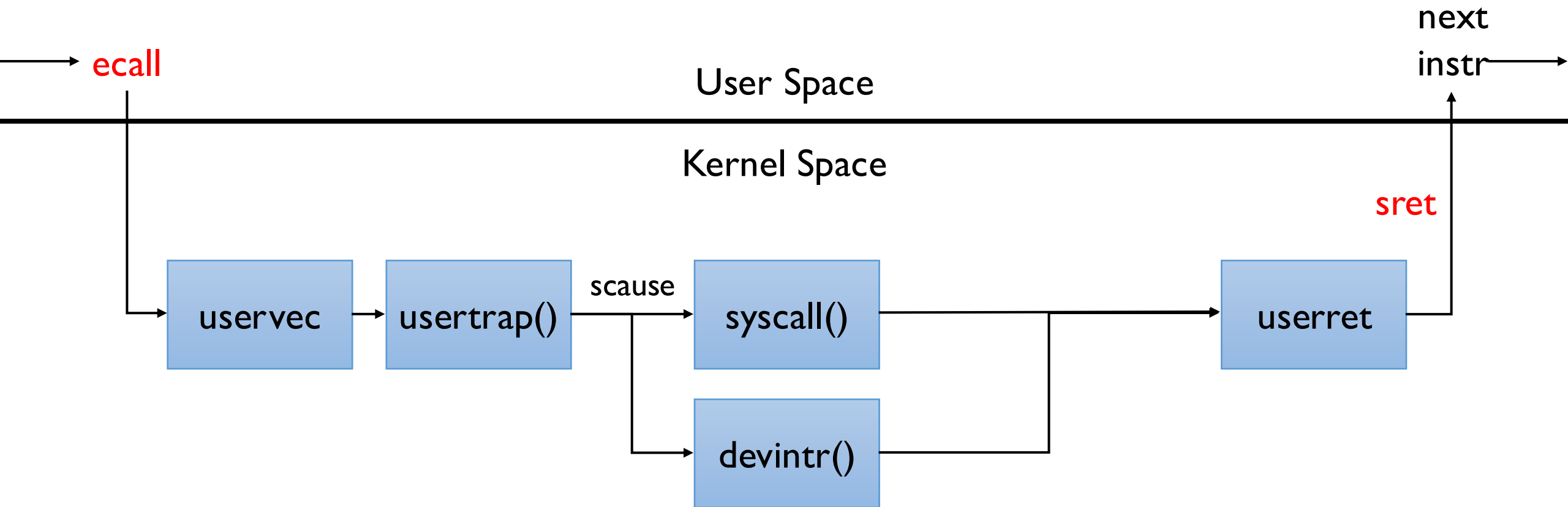
- User processes run in user mode

# ecall

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



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



# Some Registers

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

# What happens on **trap**

- The RISC-V hart performs all these steps as a **single** operation
  - Store `sstatus.SIE` to `sstatus.SPIE`
  - Disable interrupts by setting `sstatus.SIE` to 0
  - Store the originated privilege mode (U mode: 0, otherwise, I) into `sstatus.SPP`
  - Copy the `pc` into `sepc`
  - Set `scause` to reflect the trap's cause
  - Set the `stval` if necessary (e.g., fault address)
  - Copy **`stvec`(which is `uservec` in `xv6`)** to the `pc`
  - Start executing at the new `pc`

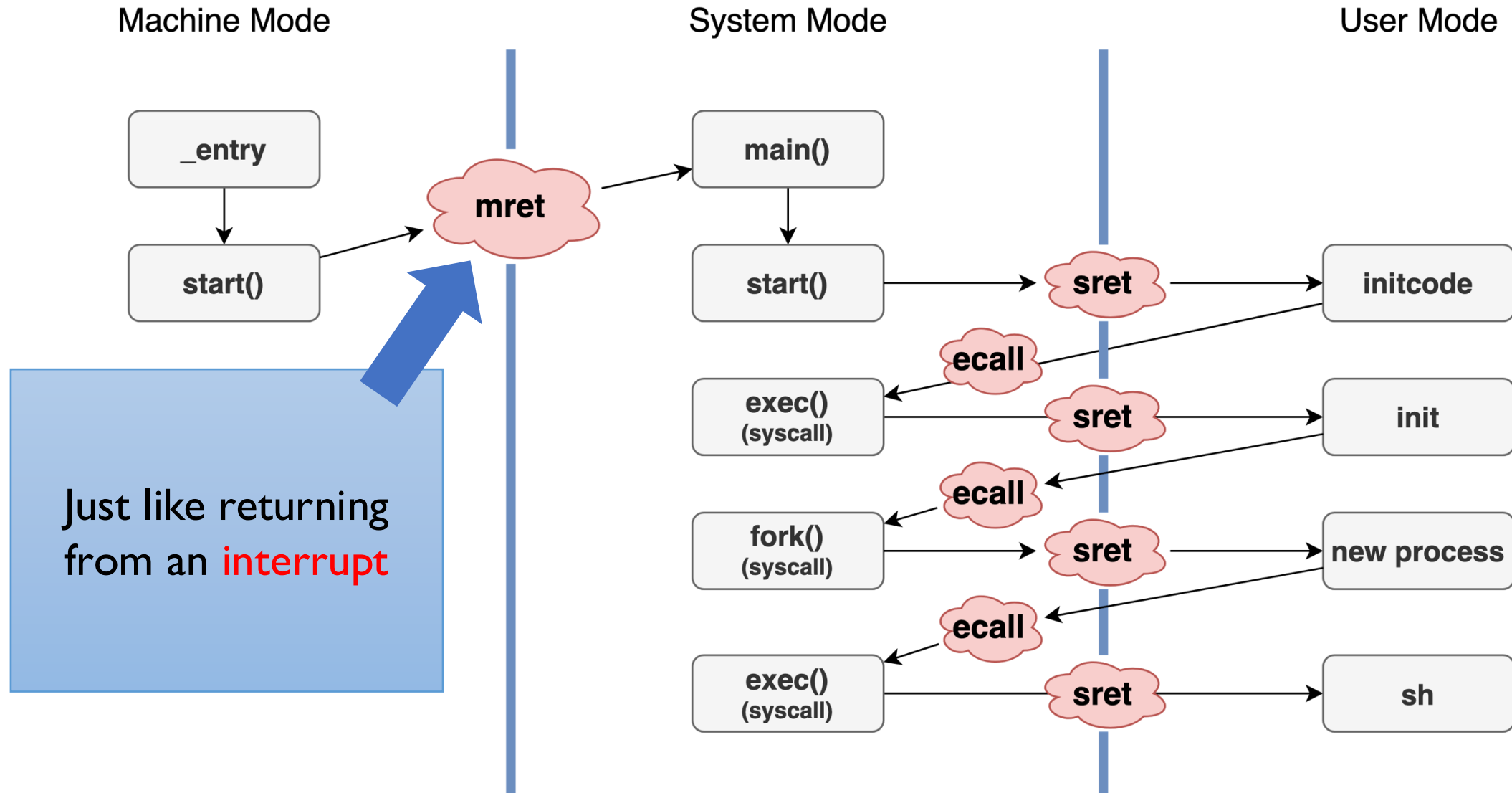
# What happens on **sret**

- The RISC-V hart performs all these steps as a **single** operation
  - Restore `sstatus.SIE` from `sstatus.SPIE`
  - Restore the originated privilege mode from `sstatus.SPP`
  - Enable interrupts by setting `sstatus.SIE` to 1
  - Copy the `sepc` into `pc`
  - Start executing at the new `pc`

# Trap Delegation

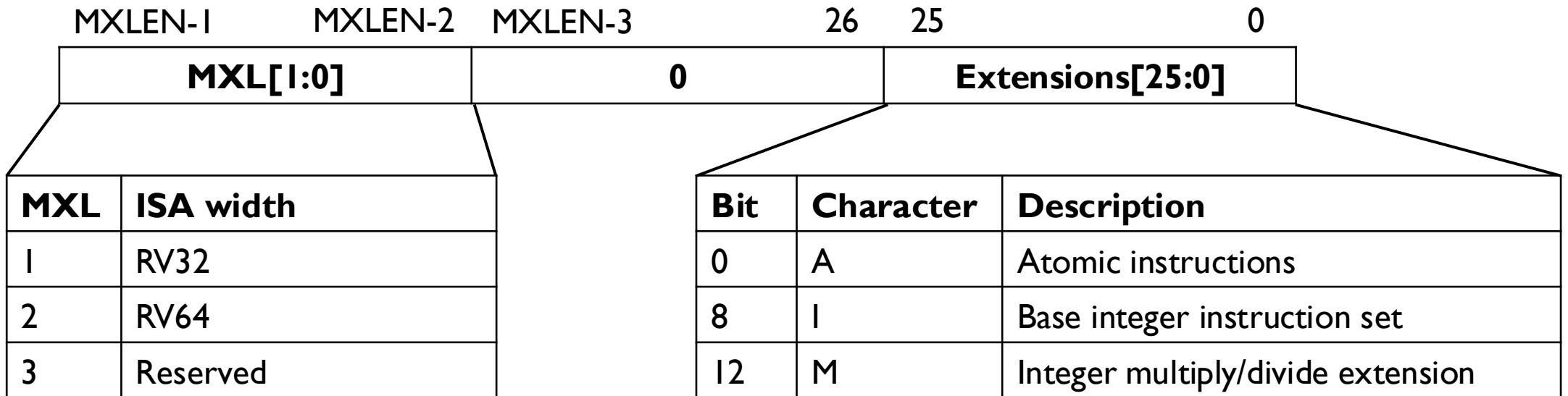
- By default, all traps are handled in S-mode
- Register **medeleg** and **mideleg** can set certain traps to be processed directly by a lower privilege level (S-mode)
- Setting a bit in **medeleg** or **mideleg** will delegate the corresponding trap, when occurring in S-mode or U-mode, to the **M-mode**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

# xv6 booting



# misal (machine ISA) register

- Machine-level control and status register(CSR) in RISC-V architecture
- Read-write register reporting the base integer width of the processor (e.g. RV32 or RV64) and the set of enabled ISA extensions



# System Reset

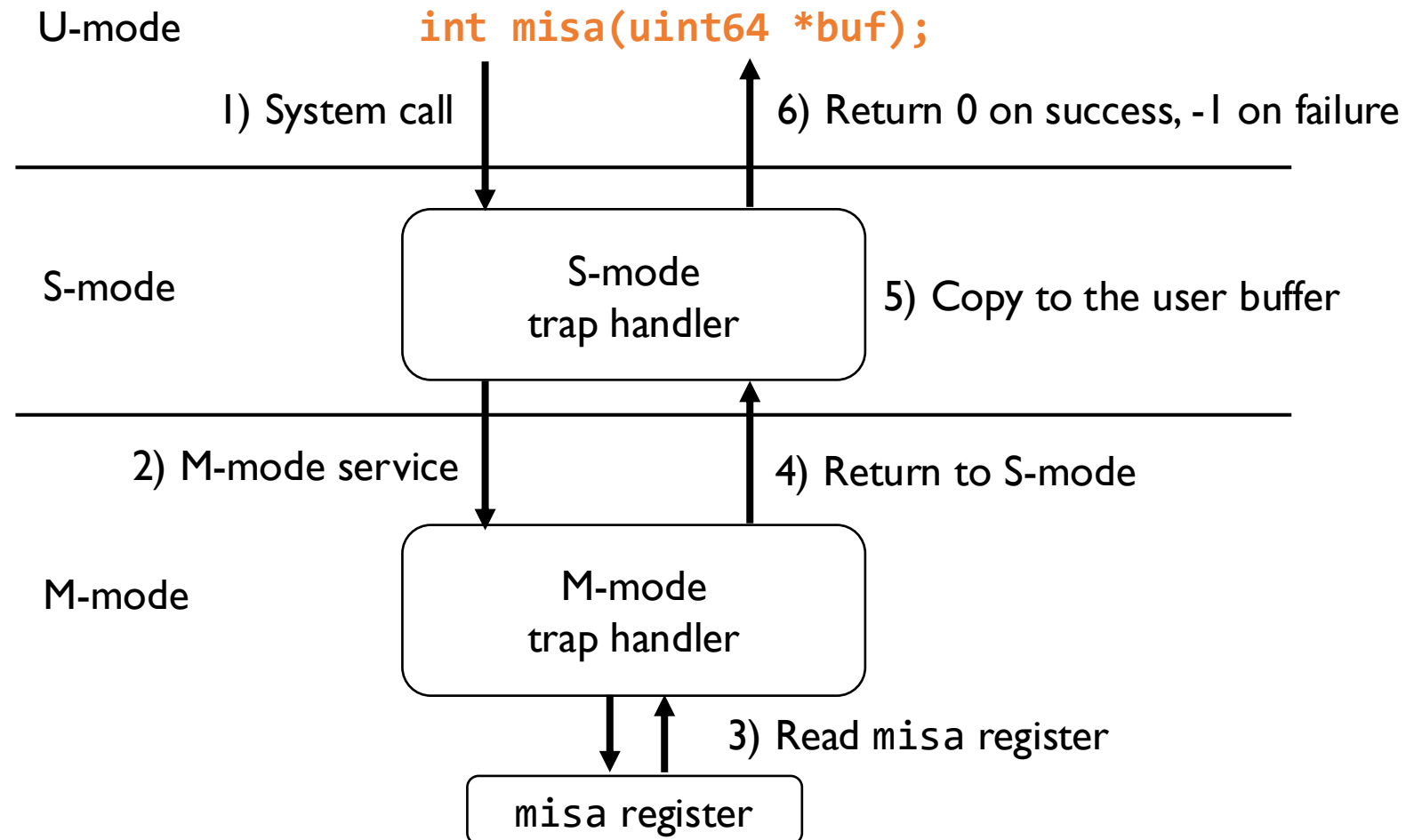
- QEMU's RISC-V virt machine emulates SiFive Test device (finisher)
- SiFive Test device
  - Located at physical address `0x100000`
  - Can be accessed by memory-mapped I/O store
  - Guest can write 32-bit value `0x7777` to the device register (`0x100000`) for `system reset` request

# Project 2

- Your task is to implement `misa()` and `reset()` system calls
- The system call number of `misa()`, `reset()` is already assigned to 22, 23 in the `kernel/syscall.h` file
- You can access `misa` register and finisher device **only in M-mode**
- Implement **privilege level transition** between M-mode and S-mode

# Project #2-1 `misa()` system call (60 points)

- Return the value of the RISC-V `misa` register

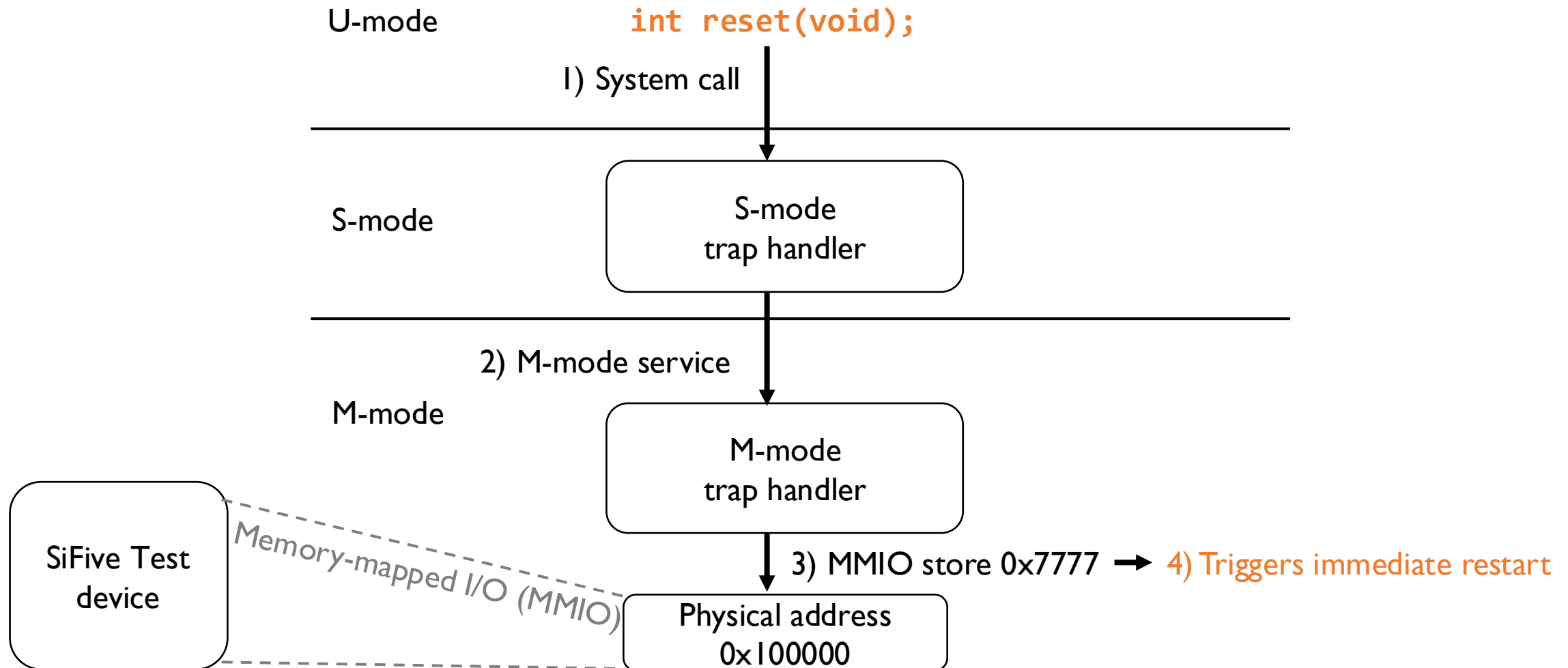


# Project #2-1 `misa()` system call (60 points)

- Retrieve the current value of `misa` without modification
- On success
  - Return 0
  - Store the `misa` register value in argument `*buf`
- On failure
  - ex) `*buf` was not a valid writable user pointer
  - Return -1
  - Does not guarantee any valid value in `*buf`

# Project #2-2 reset() system call (40 points)

- Reset the machine through the SiFive Test device (finisher device)



# Project #2-2 reset() system call (40 points)

- QEMU's finisher device address is already defined in `memlayout.h`;

```
#define FINISHER 0x100000L
```

- **On success**
  - Machine immediately resets
  - Does not return to the calling process
- **On failure**
  - Return -1

# Restriction

- You are allowed to modify only these files
  - kernel/riscv.h
  - kernel/start.c
  - kernel/machinevec.S
  - kernel/sysproc.c
  - Even if you modify other files, those changes will not be committed on the grading server
- qemu version 8.2.0 or later (`$ qemu-system-riscv64 --version`)
- Do not change the system call number (22, 23) for `misa()`, `reset()`

# Submission Guidelines

- "make submit" command to generate a compressed tar file named xv6-`{PANUM}-{STUDENTID}.tar.gz` in the `../xv6-riscv-snu` directory
- You need to submit **a report (Design Document)** to server
- Up to 30 submissions are permitted
- Only the version marked FINAL will be considered for the project score

# Due date

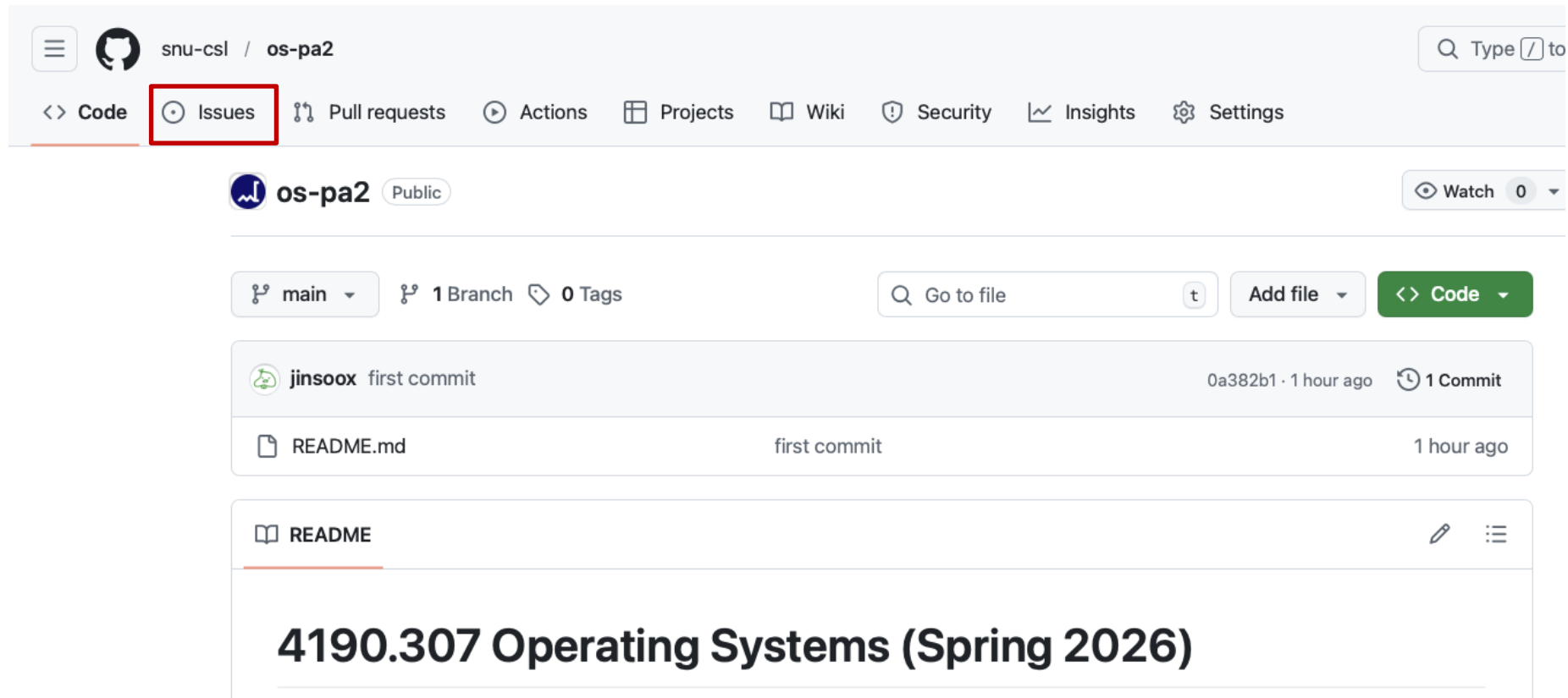
- **Due: 11:59 PM, April 5 (Sunday)**
- Only the upload submitted before the deadline will receive the full credit
- 25% of the credit will be deducted for every single day delayed

# Slip Days Policy

- You can use up to **3 slip days during this semester**
  - You should explicitly declare the number of slip days you want to use on the QnA board of the submission server **before the next project assignment is announced**
  - Once slip days have been used, they cannot be canceled later
- Submissions more than 3 days past the due date **will not be accepted**
- We highly recommend for you to save slip days for later projects!

# Q&A

- Please post project-related questions on the *Issues* tab of <https://github.com/snu-csl/os-pa2>
- Do not upload code



The screenshot shows the GitHub interface for the repository 'snu-csl / os-pa2'. The 'Issues' tab is highlighted with a red box. The repository is public and has 1 branch (main) and 0 tags. A commit by 'jinsoox' is shown with a README.md file. The README content is '4190.307 Operating Systems (Spring 2026)'.

# Project #2 repo

- **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
```

- \$ make qemu makes “fs.img” for user programs
  - You can run user program on xv6 shell

# Project #2 repo

- User-level utility programs
  - The pa2 branch supports user-level utility programs named **misa**, **reset** which can be built from the [user/misa.c](#), [user/reset.c](#) file

```
hart 2 starting
hart 1 starting
init: starting sh
$ misa
misa: 80000000001411AD A CD F HI M S U (64-bit)
$ reset
```

```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$
```

# Using GDB with QEMU

# GDB with QEMU (Linux)

- Run `sudo apt install gdb-multiarch`
- In the `xv6-riscv-snu` directory, run `make qemu-gdb` to run QEMU
- In another shell, run `gdb-multiarch ./kernel/kernel`

```
csl@sys.snu.ac.kr
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000
```

```
csl@sys.snu.ac.kr
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=riscv64-unknown-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined
by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
    add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
    set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
    info "(gdb)Auto-loading safe path"
(gdb)
```

# GDB with QEMU (Linux)

- In GDB, enter target remote :<port>
- You can find TCP port in the QEMU log

```
csl@sys.snu.ac.kr x + v - □ ×
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000
```

```
csl@sys.snu.ac.kr x + v - □ ×
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined
by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
  add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
  set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000000100 in ?? ()
(gdb)
```

# GDB with QEMU (MacOS)

- In the xv6-riscv-snu directory, run `make qemu-gdb` to run QEMU
- In another shell, run `lldb ./kernel/kernel`

```
make qemu-gdb ㄿ#1
[15:57:29] injae:xv6-riscv-snu <riscv> $ make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp
3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,f
ormat=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb
tcp::25501
```

```
lldb ./kernel/kernel ㄿ#2
[15:57:39] injae:xv6-riscv-snu <riscv> $ lldb ./kernel/kernel
(lldb) target create "./kernel/kernel"
Current executable set to '/Users/injae/xv6-riscv-snu/kernel/kernel' (riscv64).
(lldb)
```

# GDB with QEMU (MacOS)

- In LLDB, enter gdb-remote <port>
- You can find TCP port in the QEMU log

```
make qemu-gdb ㄿ#1
[15:57:29] injae:xv6-riscv-snu <riscv> $ make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp
3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,f
ormat=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb
tcp::25501
```

```
lldb ./kernel/kernel ㄿ#2
[15:57:39] injae:xv6-riscv-snu <riscv> $ lldb ./kernel/kernel
(lldb) target create "./kernel/kernel"
Current executable set to '/Users/injae/xv6-riscv-snu/kernel/kernel' (riscv64).
(lldb) gdb-remote 25501
Process 1 stopped
* thread #1, stop reason = signal SIGTRAP
   frame #0: 0x0000000000000100
-> 0x1000: auipc t0, 0
   0x1004: addi a2, t0, 40
   0x1008: csrr a0, mhartid
   0x100c: ld a1, 32(t0)
Target 0: (kernel) stopped.
(lldb) █
```

# GDB with QEMU

- The xv6 virtual machine has stopped at 0x1000 (the very beginning of the text section)
- To execute shell, enter c in GDB

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ █
```

(Running)

```
csl@sys.snu.ac.kr x + v
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined
by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
  add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
  set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x00000000000001000 in ?? ()
(gdb) c
Continuing.
█
```

# GDB with QEMU

- To stop again, enter Ctrl-C in GDB
  - Cannot input command to shell
- Then the xv6 virtual machine stops immediately

```
csl@sys.snu.ac.kr
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nog
raphic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id
=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ █

(Stopped)
```

```
csl@sys.snu.ac.kr
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined
by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
    add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
    set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000001000 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
mycpu () at kernel/proc.c:79
79     {
(gdb) █
```

# GDB with QEMU

- Let's set a breakpoint at exec()
- Enter b exec in GDB

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ █
```

(Stopped)

```
csl@sys.snu.ac.kr x + v
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined
by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
  add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
  set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000001000 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
mycpu () at kernel/proc.c:79
79   {
(gdb) b exec
Breakpoint 1 at 0x80004ec0: file kernel/exec.c, line 24.
(gdb) █
```

# GDB with QEMU

- Enter c in GDB to resume the xv6 machine

```
csl@sys.snu.ac.kr
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$
```

(Running)

```
csl@sys.snu.ac.kr
by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
    add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
    set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
    info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000001000 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
mycpu () at kernel/proc.c:79
79     {
(gdb) b exec
Breakpoint 1 at 0x80004ec0: file kernel/exec.c, line 24.
(gdb) c
Continuing.
█
```

# GDB with QEMU

- Run ls command in the xv6 machine
- Then the xv6 machine hits the breakpoint and stops right before starting exec() function

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ ls
█
```

(Stopped)

```
csl@sys.snu.ac.kr x + v
set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual. E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000001000 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
mycpu () at kernel/proc.c:79
79      {
(gdb) b exec
Breakpoint 1 at 0x80004ec0: file kernel/exec.c, line 24.
(gdb) c
Continuing.
[Switching to Thread 1.2]

Thread 2 hit Breakpoint 1, exec (path=path@entry=0x3fffff9f00 "ls",
      argv=argv@entry=0x3fffff9e00) at kernel/exec.c:24
24      {
(gdb) █
```

# Useful GDB commands

- `info reg [register name]`
- `info thread`
  - see harts' information
- `thread [n]`
  - change hart
- `bt`
  - See call trace

# Useful LLDB commands (MacOS)

- register read [register name]
- thread list
  - see harts' information
- thread select [n]
  - change hart
- bt
  - See call trace

# More about GDB

- To learn GDB in detail, search for GDB on Google
- There are many useful videos about GDB in YouTube
- [\[JT\]의 리눅스탐험\] GDB 활용하기](#)

Thank you!