Project #2: System Calls

Sejun Kwon
(sejun000@snu.ac.kr)
Systems Software &
Architecture Lab.
Seoul National University

2025.09.25

System Calls

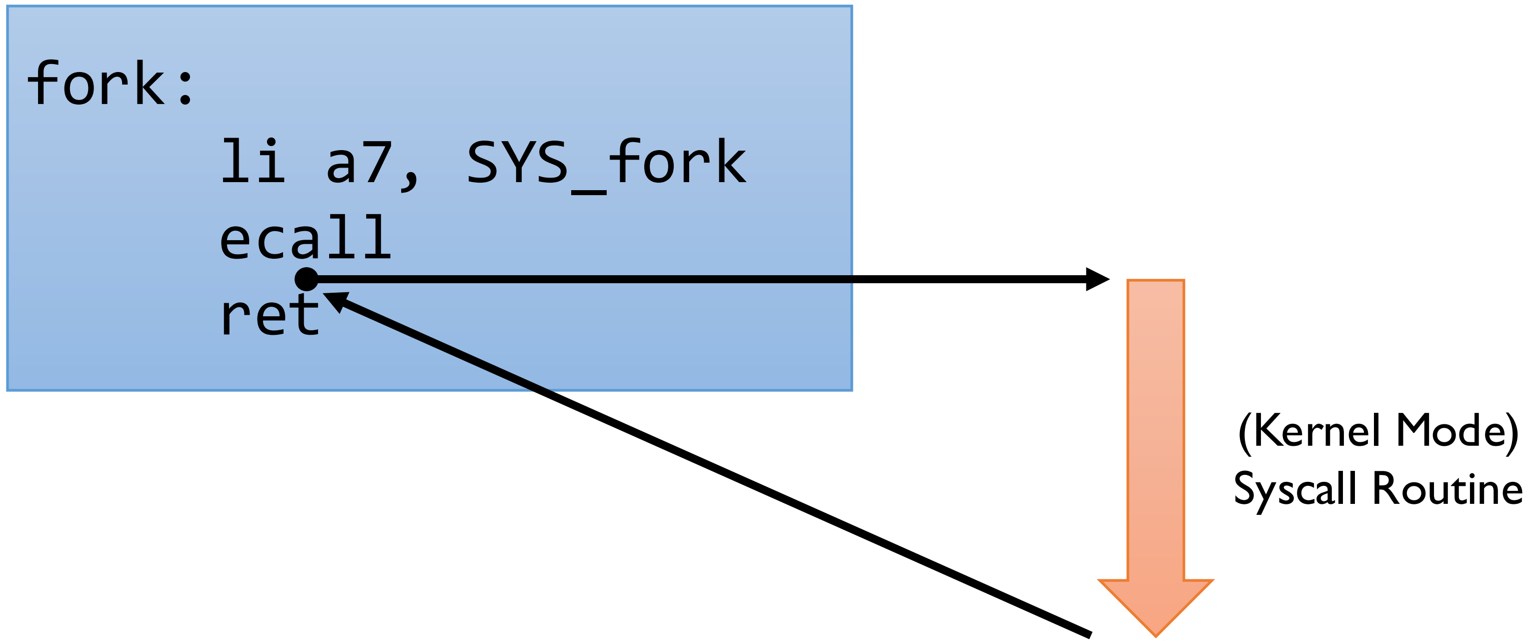
- User applications can access the operating system kernel in a restricted way
- The interfaces that allow user applications to request services from the operating system kernel
- The operating system kernel does the requested task on behalf of user applications

Three RISC-V privilege modes

- Machine Mode
  - CPU starts in machine mode
  - 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

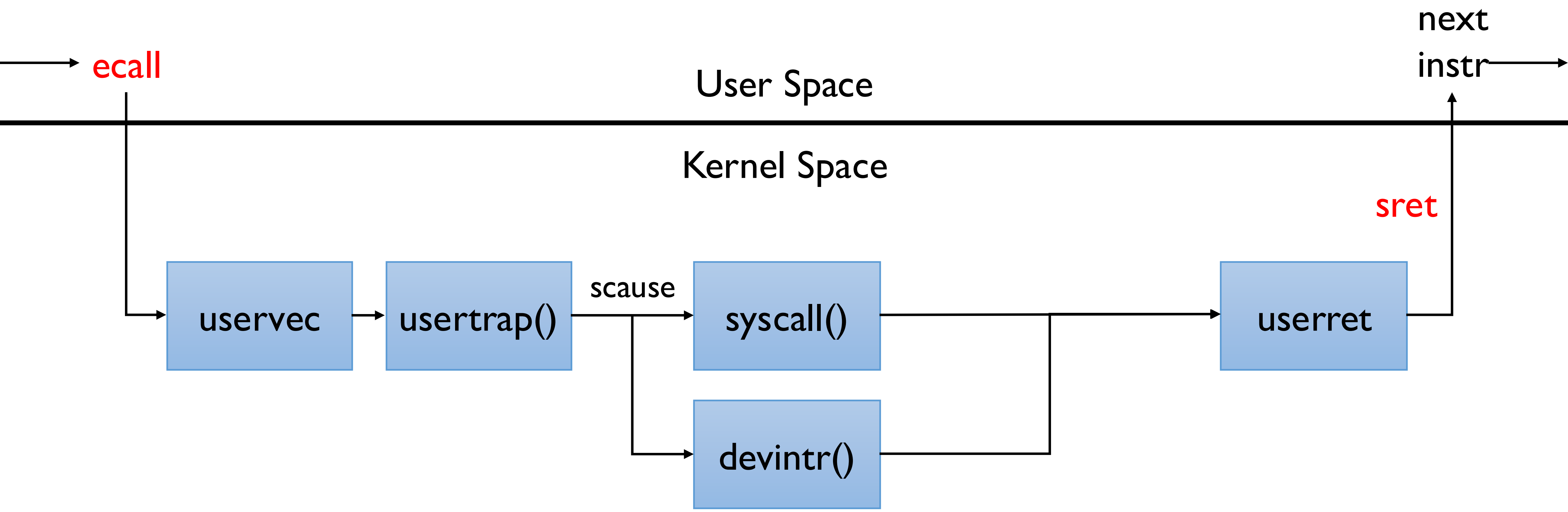
ecall

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



```
fork:
        li a7, SYS_fork
        ecall
        ret
```

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

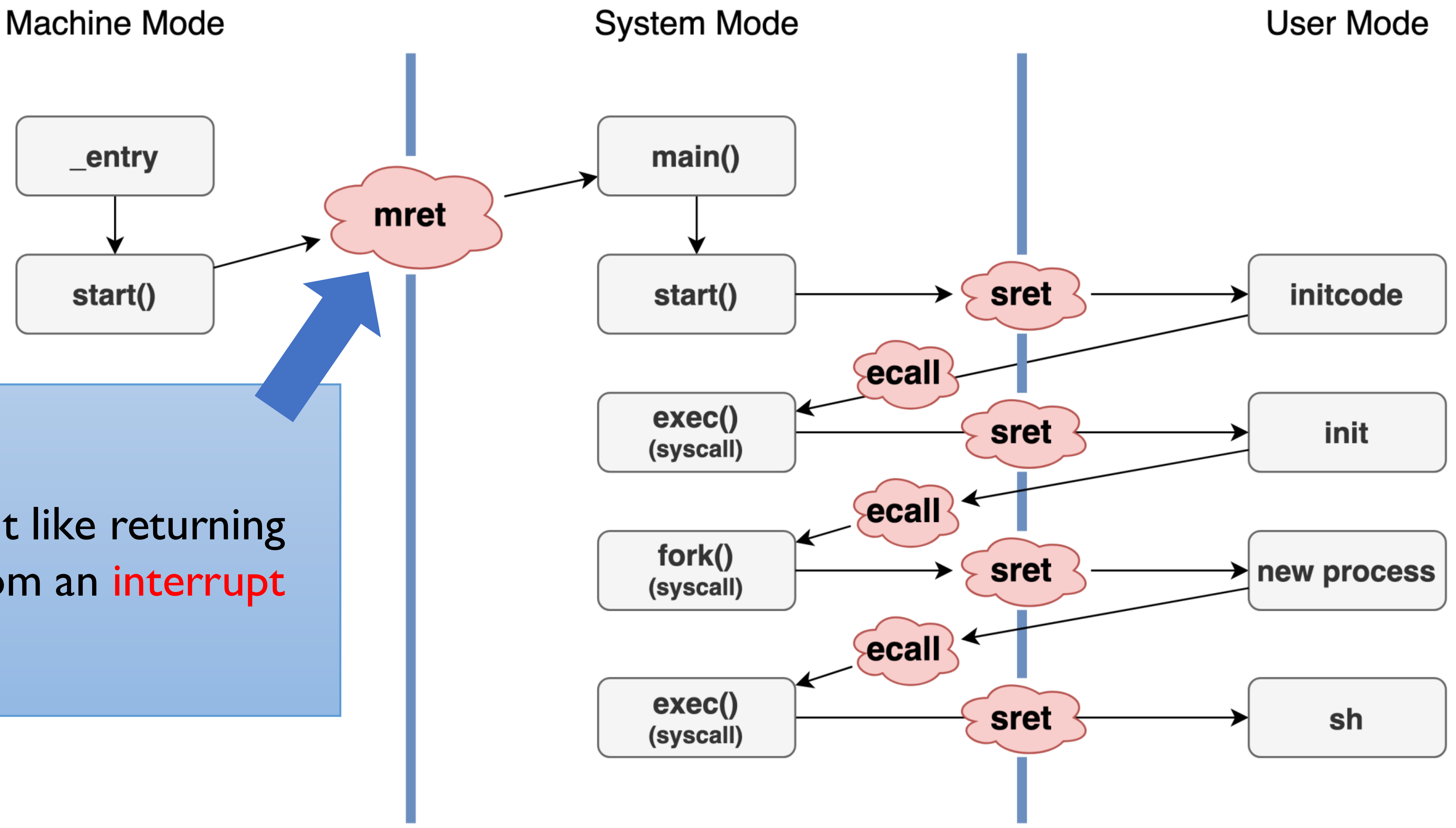
What happens on trap

- 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 current privilege mode (U) 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 current privilege mode from **`sstatus.SPP`**
  - Set **`sstatus.SPIE`** to 1
  - Copy the **`sepc`** into **`pc`**
  - Start executing at the new **`pc`**

xv6 booting



Just like returning from an interrupt

Trap delegation

- By default, all traps are handled in S-mode
- Register medeleg and mideleg can set certain traps to be processed dire ctly 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.

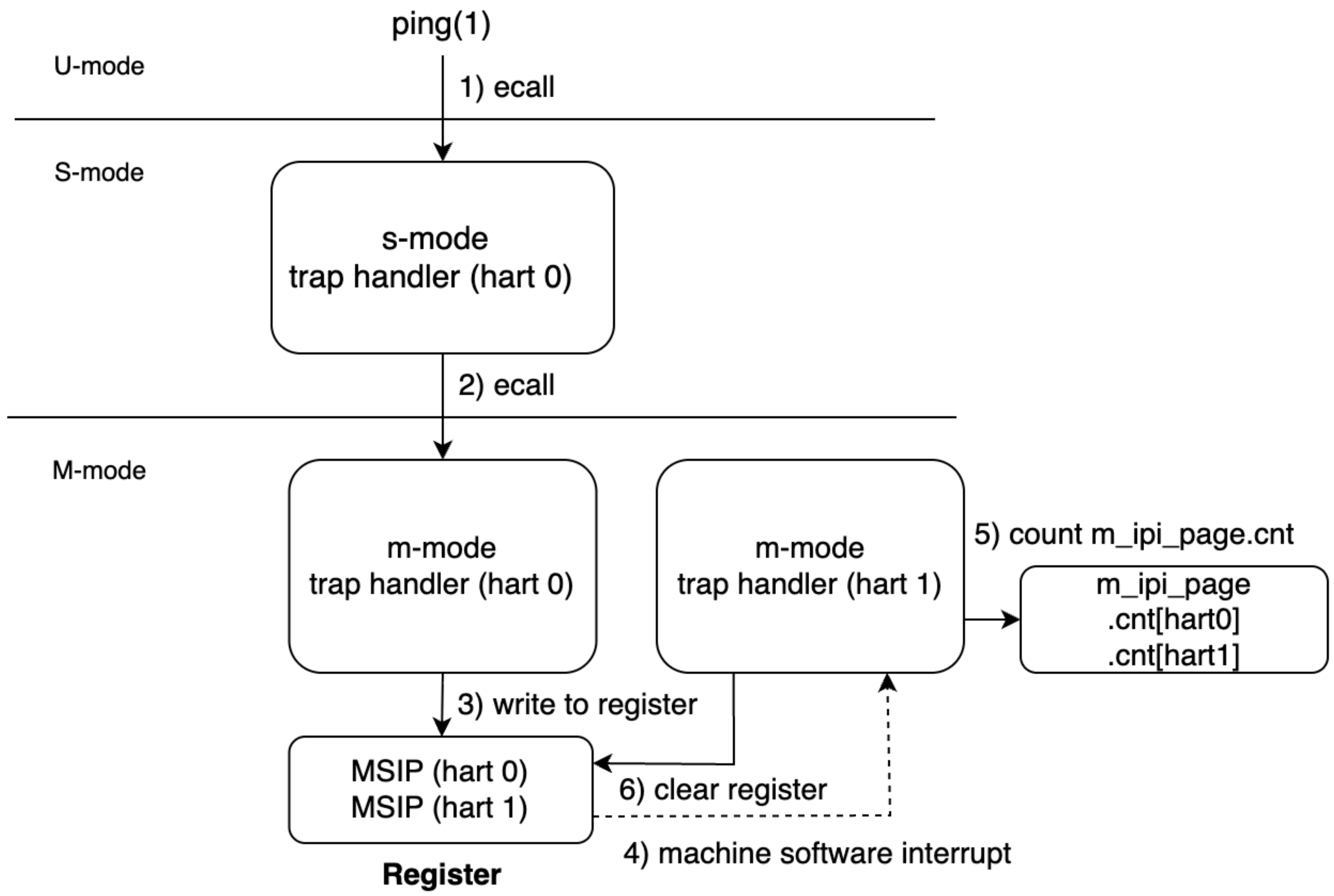
Core-Local Interrupt (CLINT)

- Local interrupts are signaled directly to an individual hart
- A software interrupt is triggered by writing 1 to the target hart's MSIP
- Since it is level-triggered, writing 1 again while it is already set does not queue another interrupt
- When MSIP is pending, and if the target hart's MIE.MSIE enabled
  - The trap is taken into M-mode and handled by the machine trap vector

Project 2

- Your task is to implement ping() and pong() system calls
- The system call number of ping(), pong() is already assigned to 22, 23 in the kernel/syscall.h file
- implement a minimal IPI mechanism in xv6 that lets one RISC-V hart simply ring a "doorbell" on another
- pong() returns the total count of ping() invocations

Project #2-1 Ping (60 points)

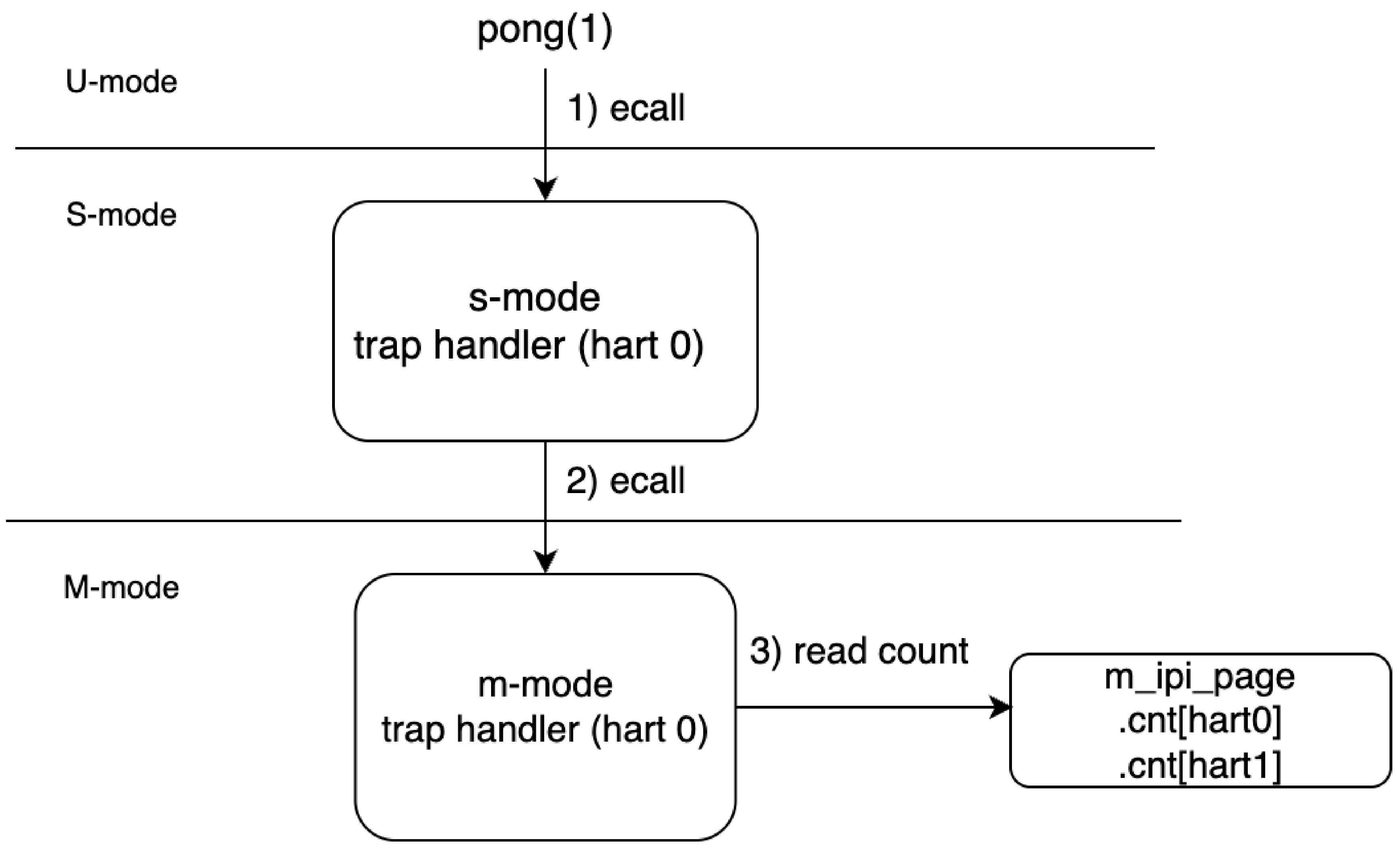


ping(1)
U-mode
1) ecall
S-mode
s-mode trap handler (hart 0)
2) ecall
M-mode
m-mode trap handler (hart 0)
m-mode trap handler (hart 1)
5) count m_ipi_page.cnt
m_ipi_page .cnt[hart0] .cnt[hart1]
3) write to register
MSIP (hart 0)
MSIP (hart 1)
6) clear register
4) machine software interrupt
Register

Project #2-1 Ping (60 points)

- Partial credit is given for each test case
  - The machine trap caused by ecall must occur exactly N times
  - The machine trap from interrupts to each hart should occur exactly N times
  - The hart receiving interrupts must deassert MSIP
  - pong() returns the total count of ping() invocations

Project #2-2 Pong (30 points)



Restriction

- You are allowed to modify only these files
  - kernel/ipi.c
  - kernel/ipi.h
  - kernel/machinevec.S
  - kernel/sysproc.c
  - Even if you modify other files, those changes will not be committed on the grading server
- The struct m_ipi_page_t definition must be modified carefully.
  - The expression "m_ipi_page.hart[1].cnt = 1" in other files must compile.
- qemu version 8.2.0 or later ($ qemu-system-riscv64 –version)
  - Do not change the system call number for ping(), pong()

Restriction

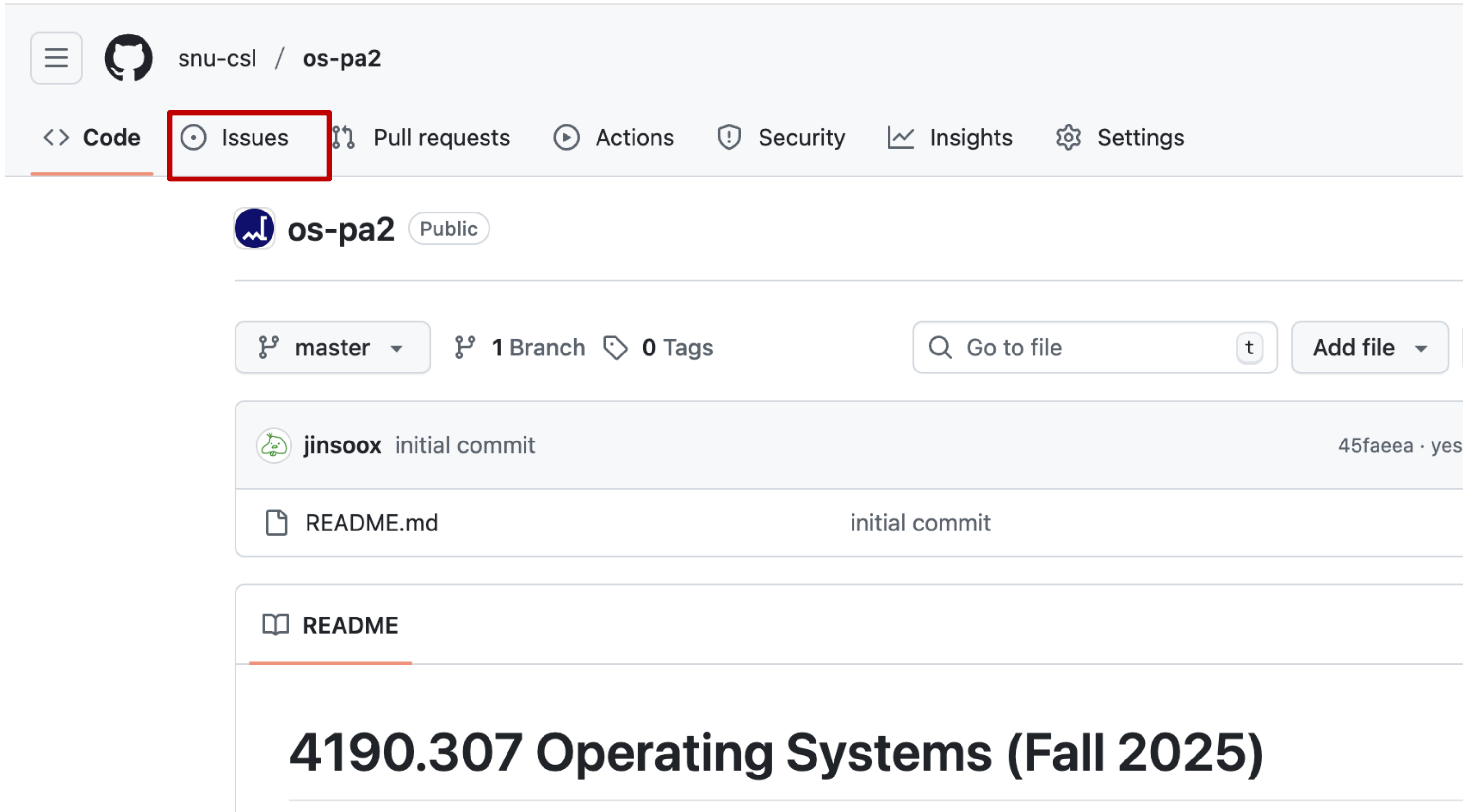
- "make submit" command to generate a compressed tar file named xv6-{PANUM}-{STUDENTID}.tar.gz in the ../xv6-riscv-snu directory
- You need to submit a report (Design Document) to server
- Up to 30 submissions are permitted
- You can use up to 3 slip days during this semester
  - You should explicitly 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
- Only the version marked FINAL will be considered for the project score

Due date

- **Due: 11:59 PM, October 12** (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.

Q&A

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



jinsoox initial commit 45faeea · yes

README.md initial commit

README

4190.307 Operating Systems (Fall 2025)

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
```

  - The pa2 branch has a user-level utility program named **ping**, **pong** which can be built from the user/ping.c, user/pong.c file

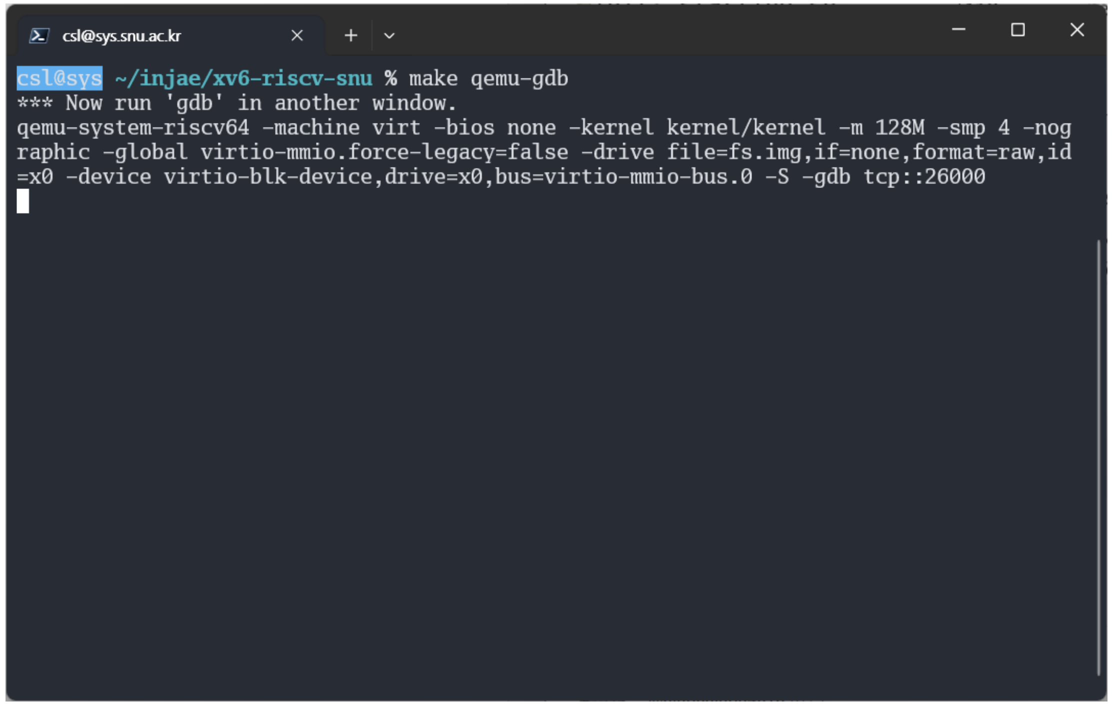
  "make qemu" makes "fs.img" for user programs

  - You can run user program on xv6 shell

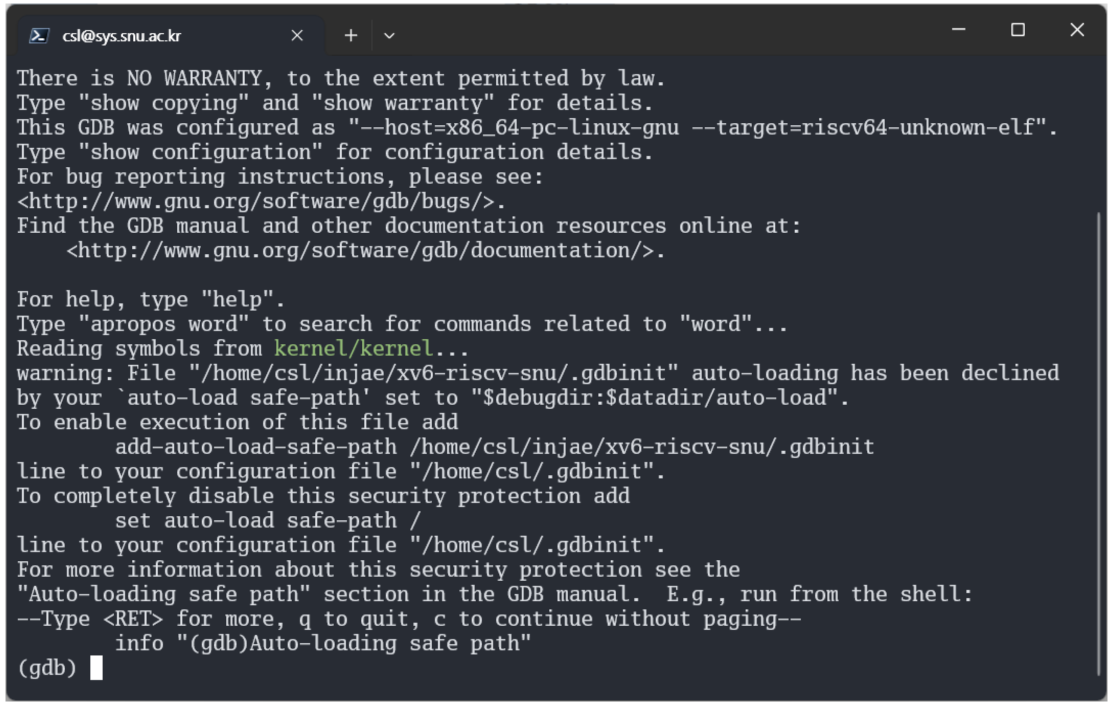
Using GDB with QEMU

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



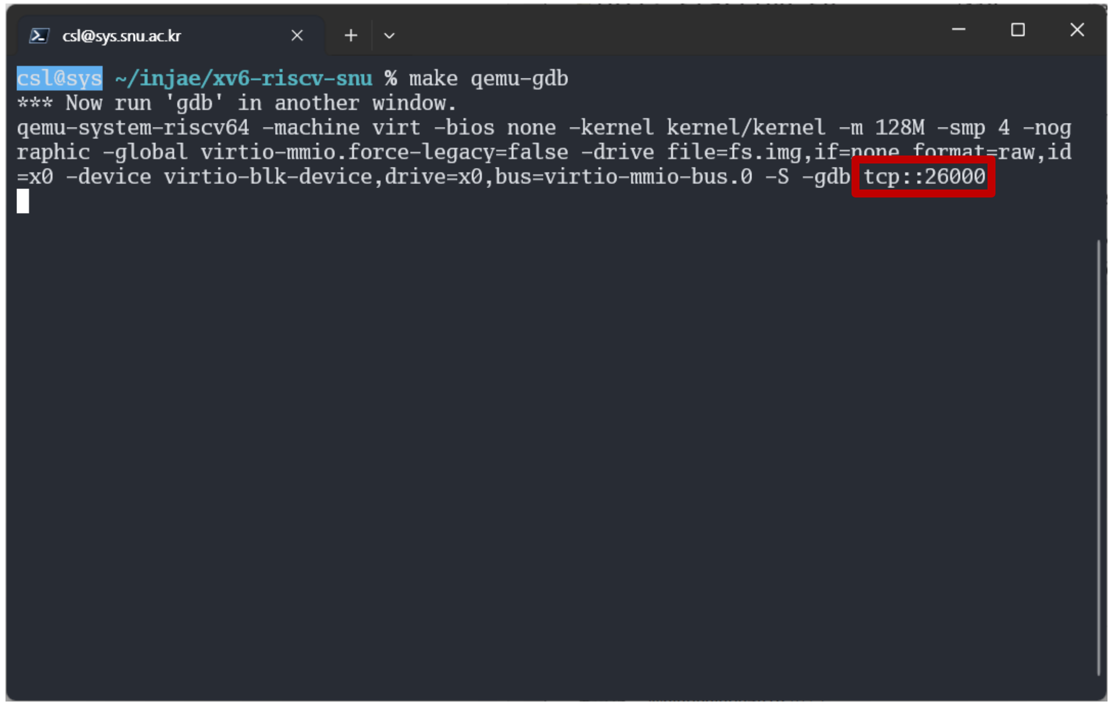
```
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nog
raphic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id
=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0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



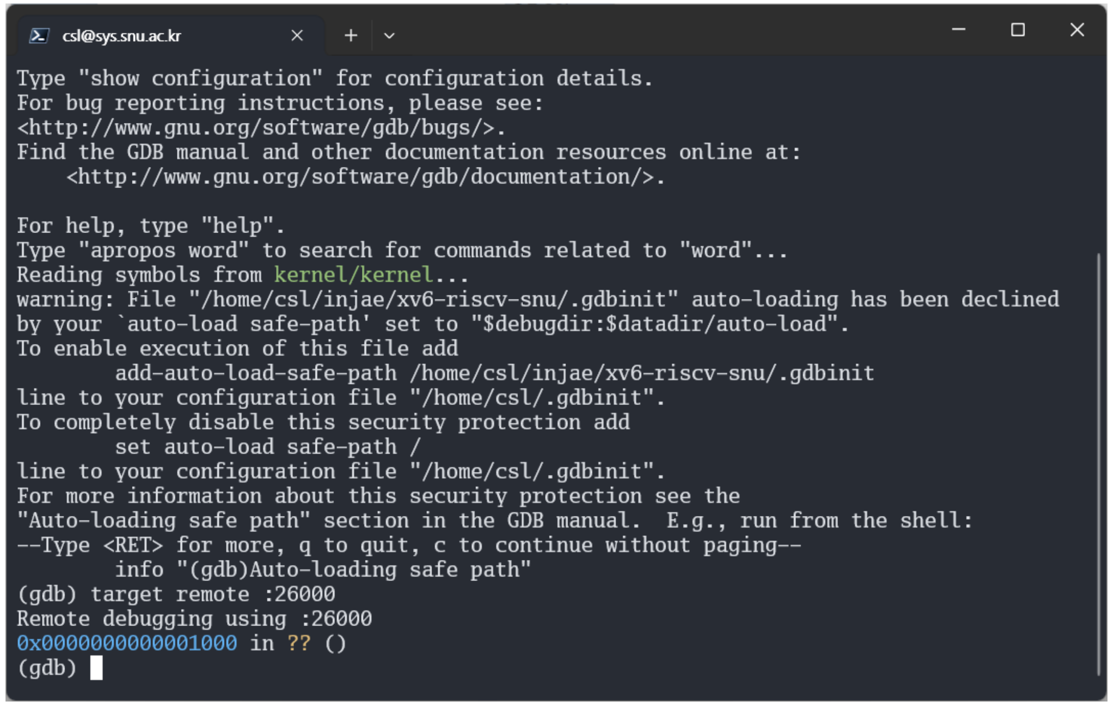
```
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=riscv64-unknown-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.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



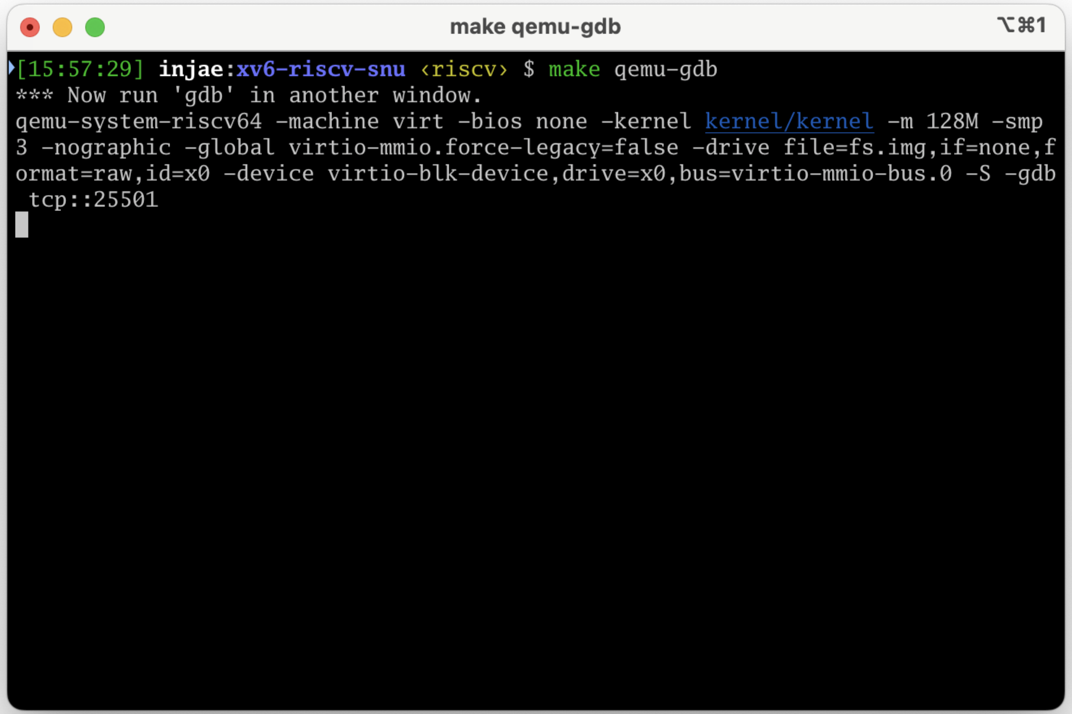
```
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nog
raphic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id
=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0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



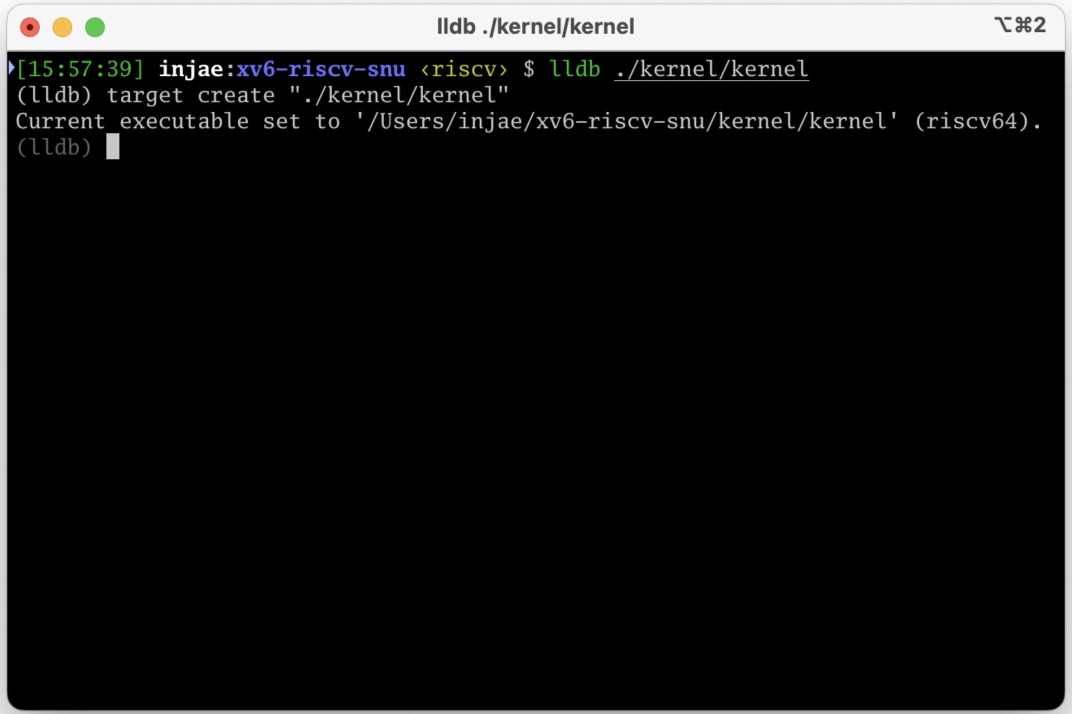
```
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

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
0x0000000000001000 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 ⌥⌘1

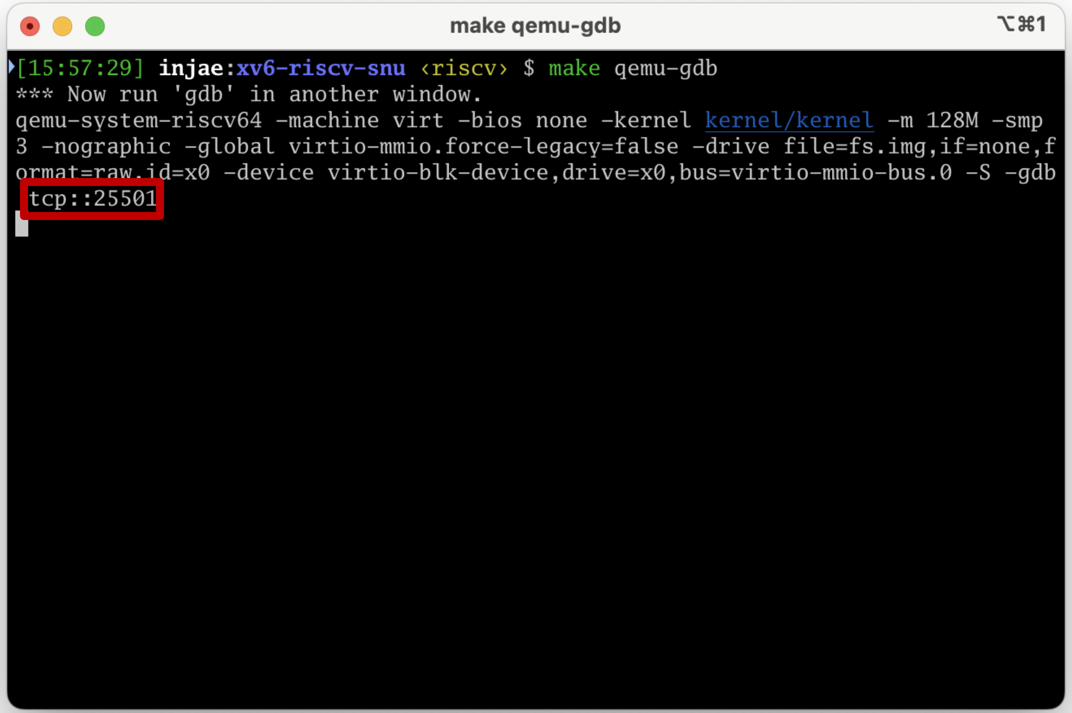
```
[15:57:29] injae:xv6-riscv-snu ‹riscv› $ make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp
3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,f
ormat=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0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


lldb ./kernel/kernel ⌥⌘2

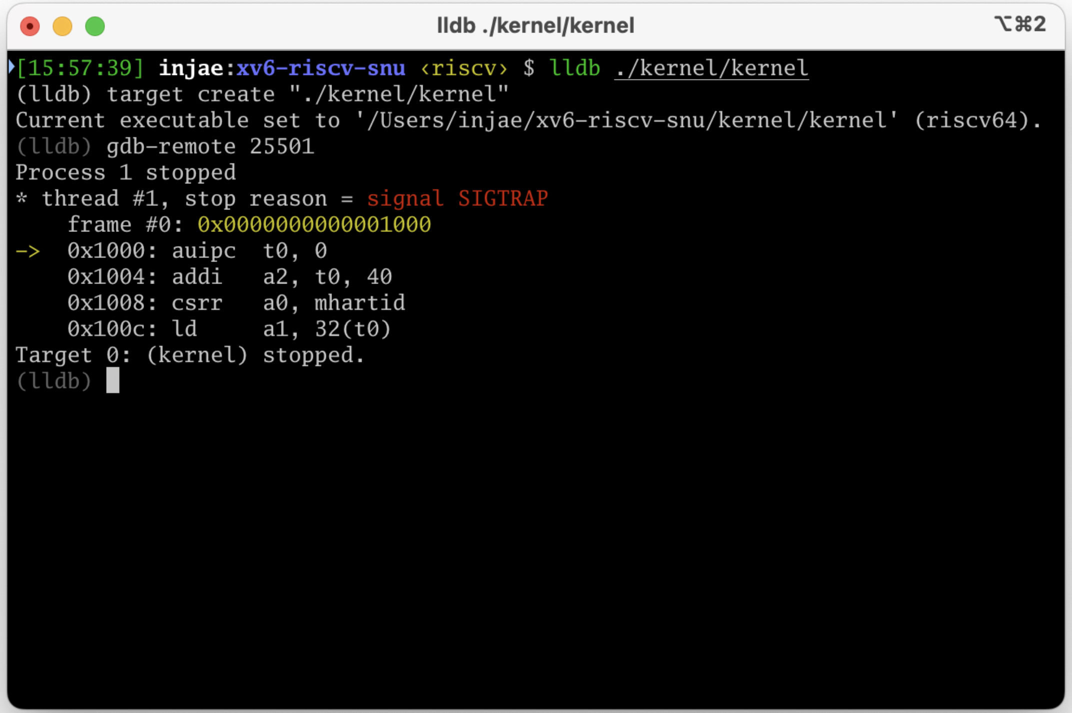
```
[15:57:39] injae:xv6-riscv-snu ‹riscv› $ lldb ./kernel/kernel
(lldb) target create "./kernel/kernel"
Current executable set to '/Users/injae/xv6-riscv-snu/kernel/kernel' (riscv64).
(lldb)
```

GDB with QEMU (MacOS)

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



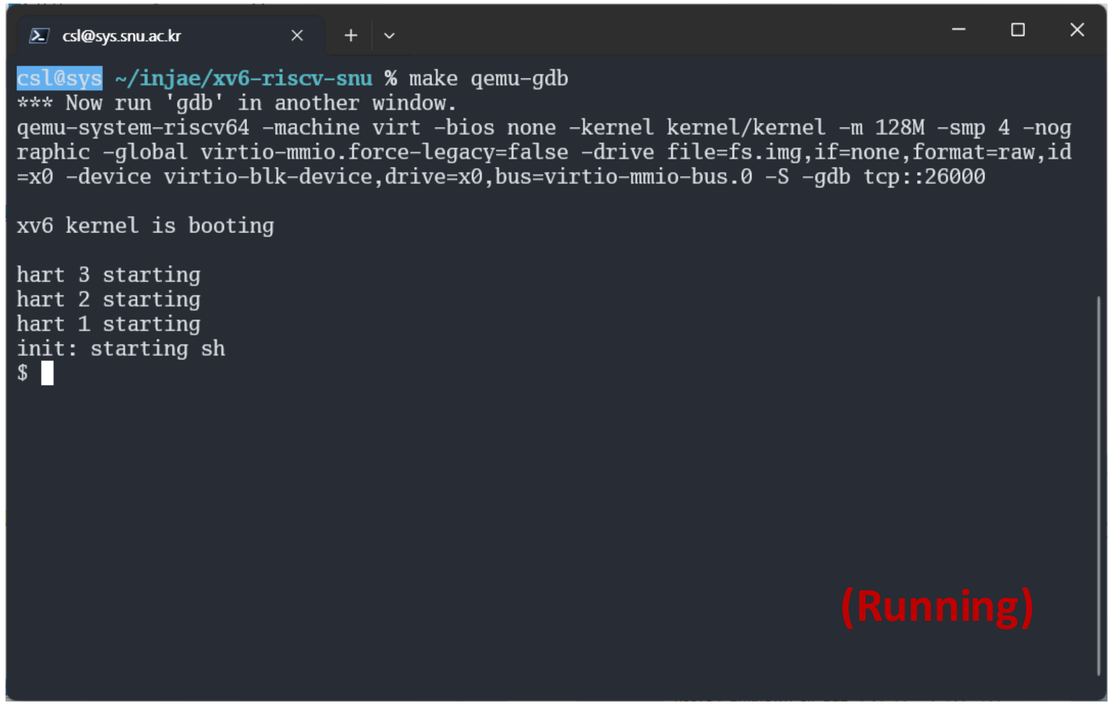
```
[15:57:29] injae:xv6-riscv-snu ‹riscv› $ make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp
3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,f
ormat=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0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



```
[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: 0x0000000000001000
->  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

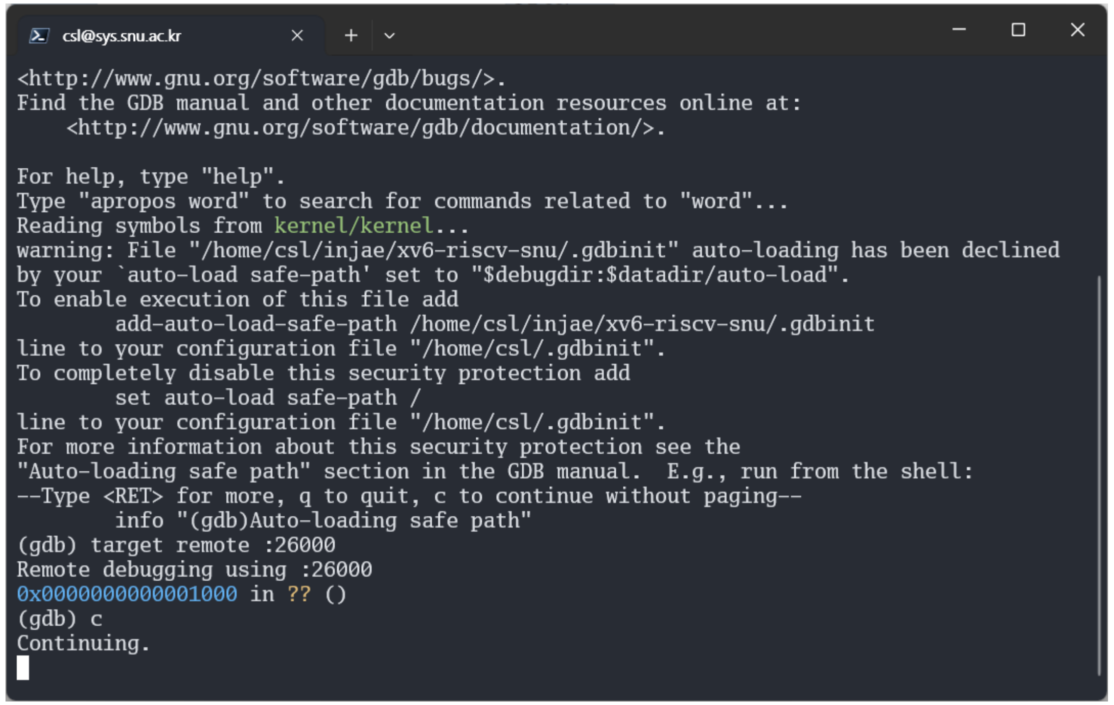
- The xv6 virtual machine has stopped at 0x1000 (the very beginning of the text section)
- To execute shell, enter `c` in GDB



csl@sys.snu.ac.kr

```
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
$ 
```

(Running)

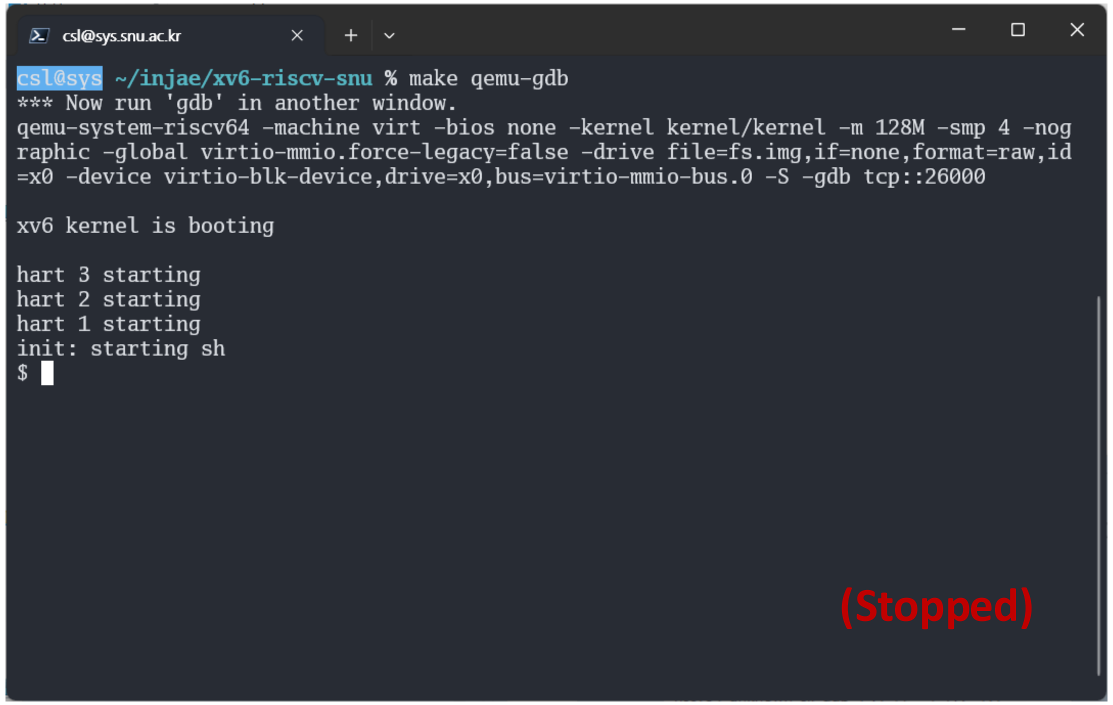


csl@sys.snu.ac.kr

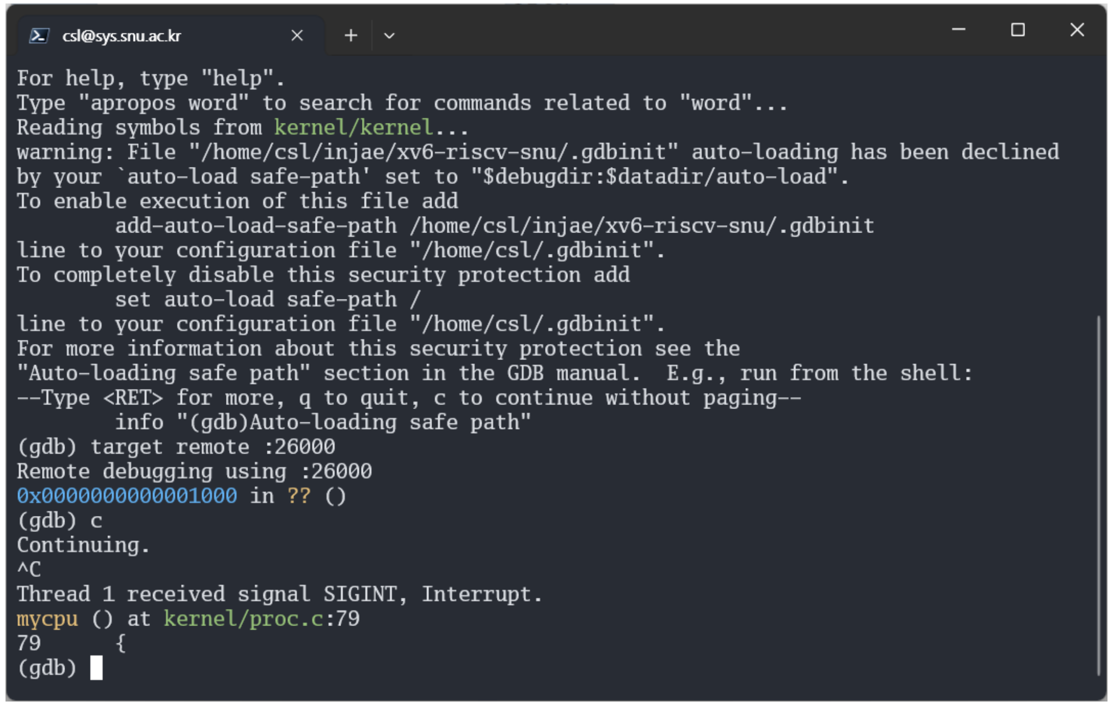
```
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

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
0x0000000000001000 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



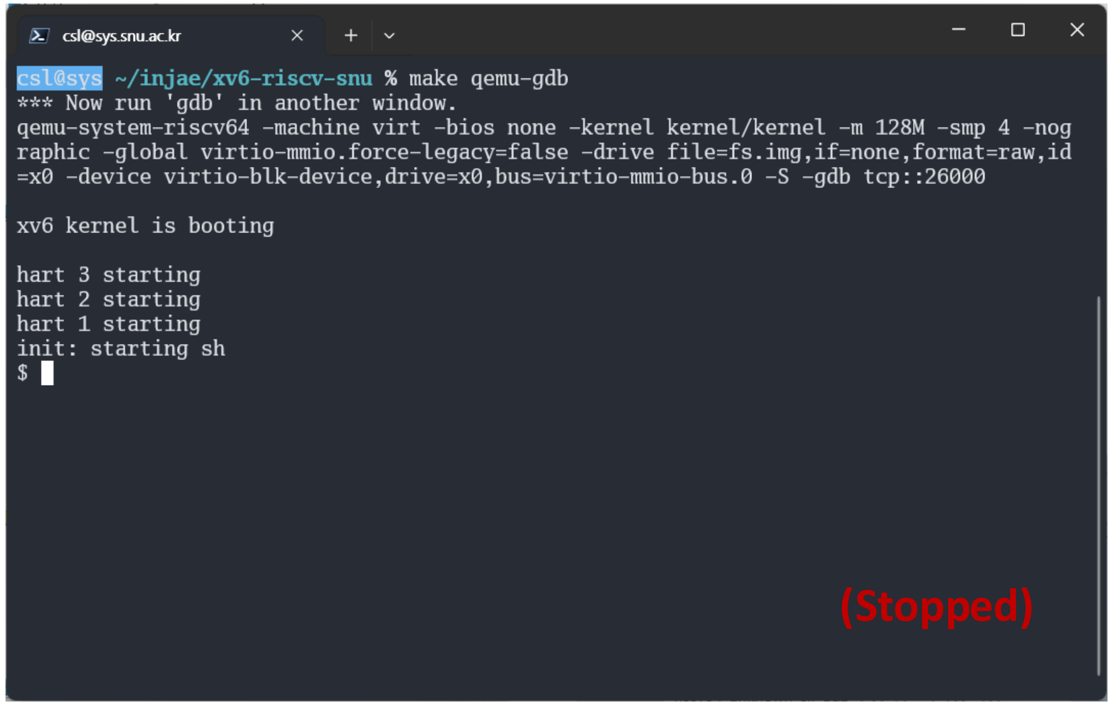
```
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nog
raphic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id
=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0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

xv6 kernel is booting

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



```
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)
```

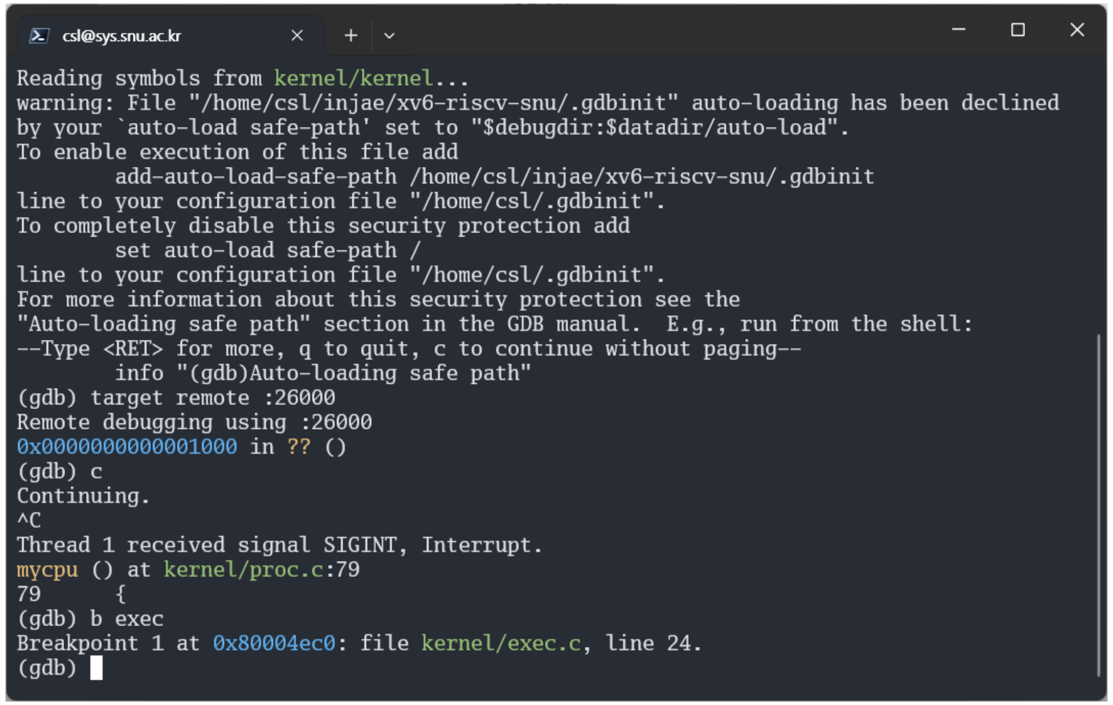
GDB with QEMU

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



```
csl@sys ~/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
$
```

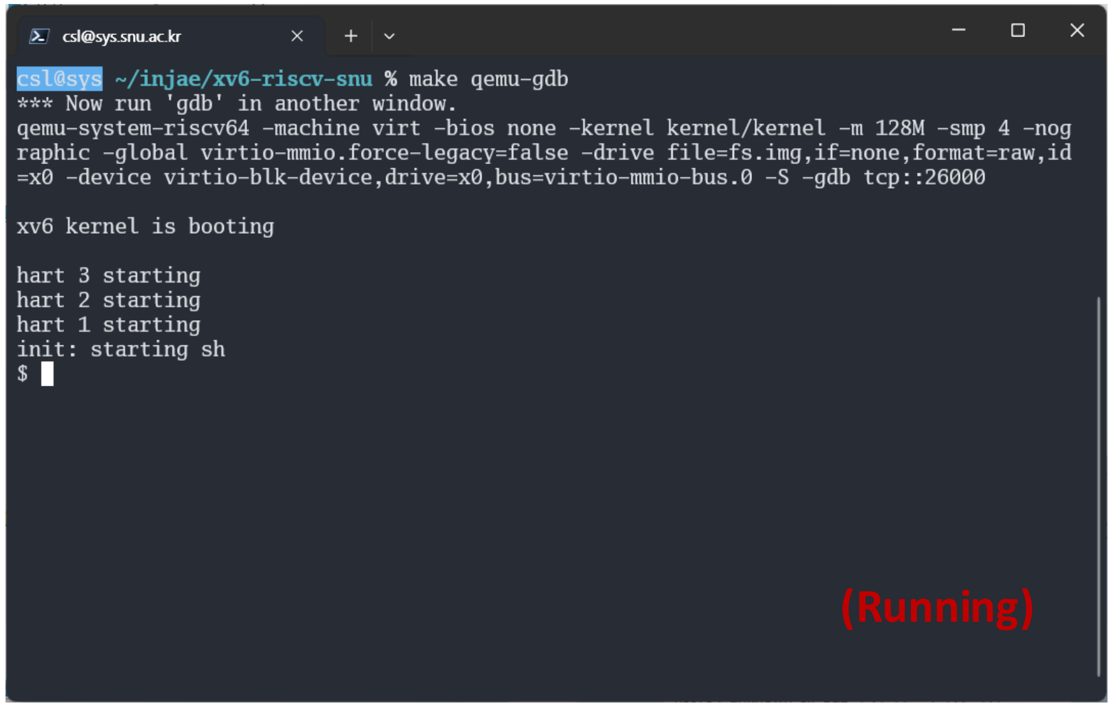
(Stopped)



```
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

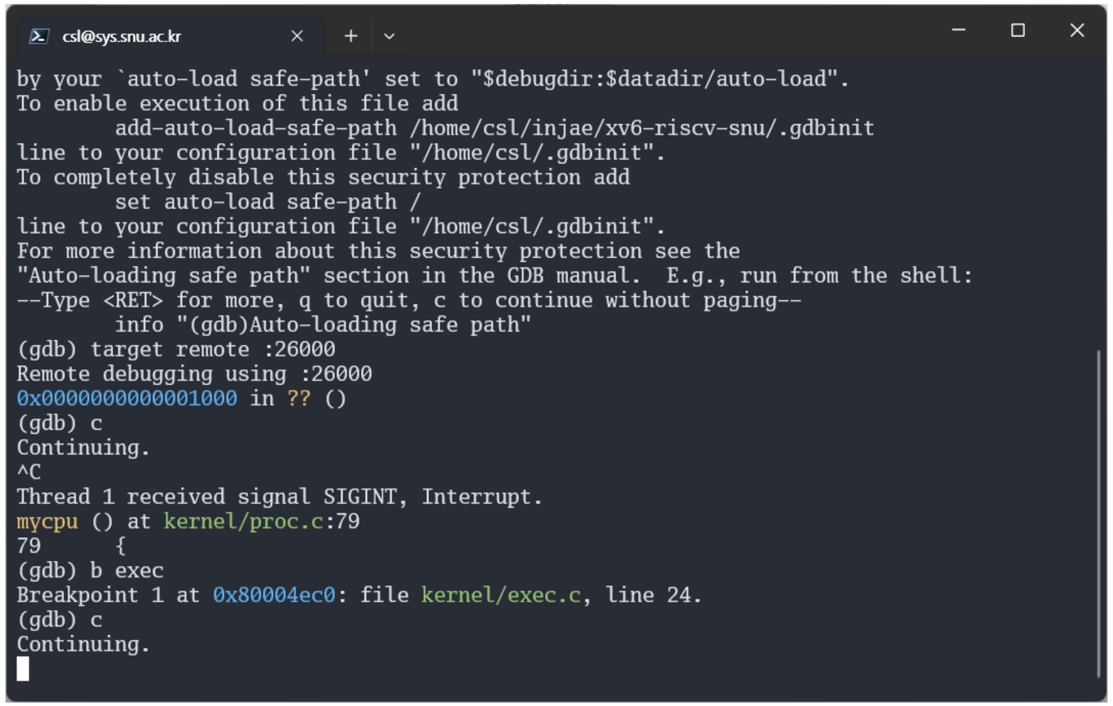
- Enter `c` in GDB to resume the xv6 machine



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 -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
$
```

(Running)

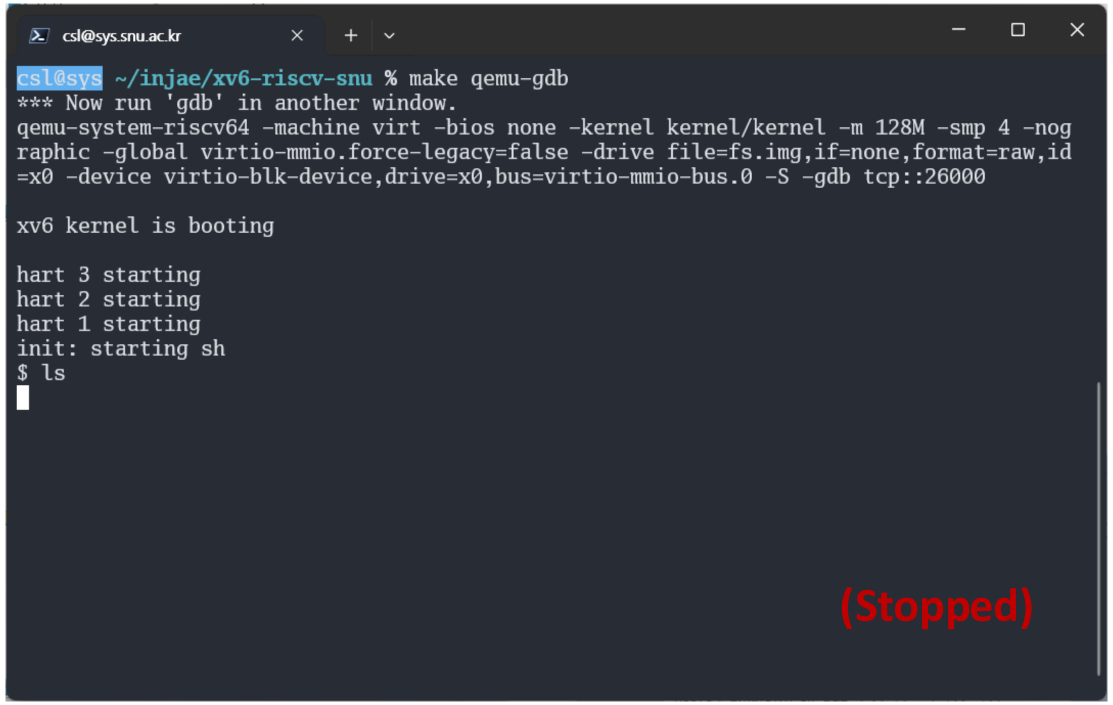


csl@sys.snu.ac.kr

```
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.
```

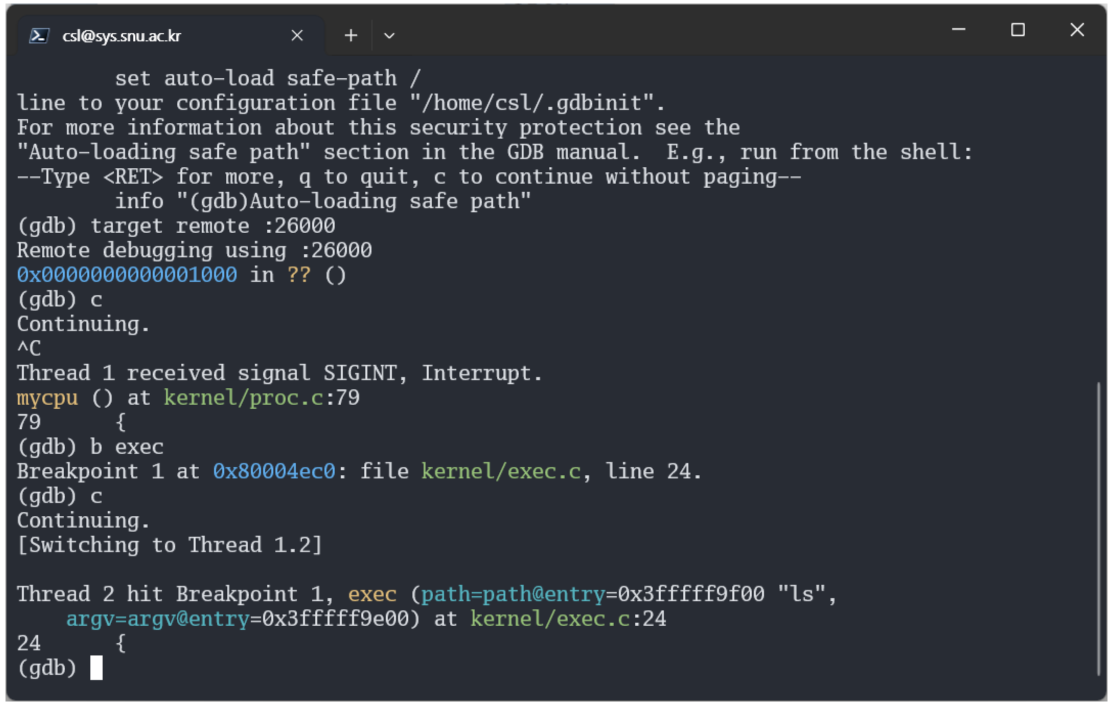
GDB with QEMU

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



```
csl@sys ~/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
$ ls
```

(Stopped)



```
        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

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!