

Injae Kang
(abcinje@snu.ac.kr)
Systems Software &
Architecture Lab.
Seoul National University

2026.09.17.

Project #2: System Calls



System Calls

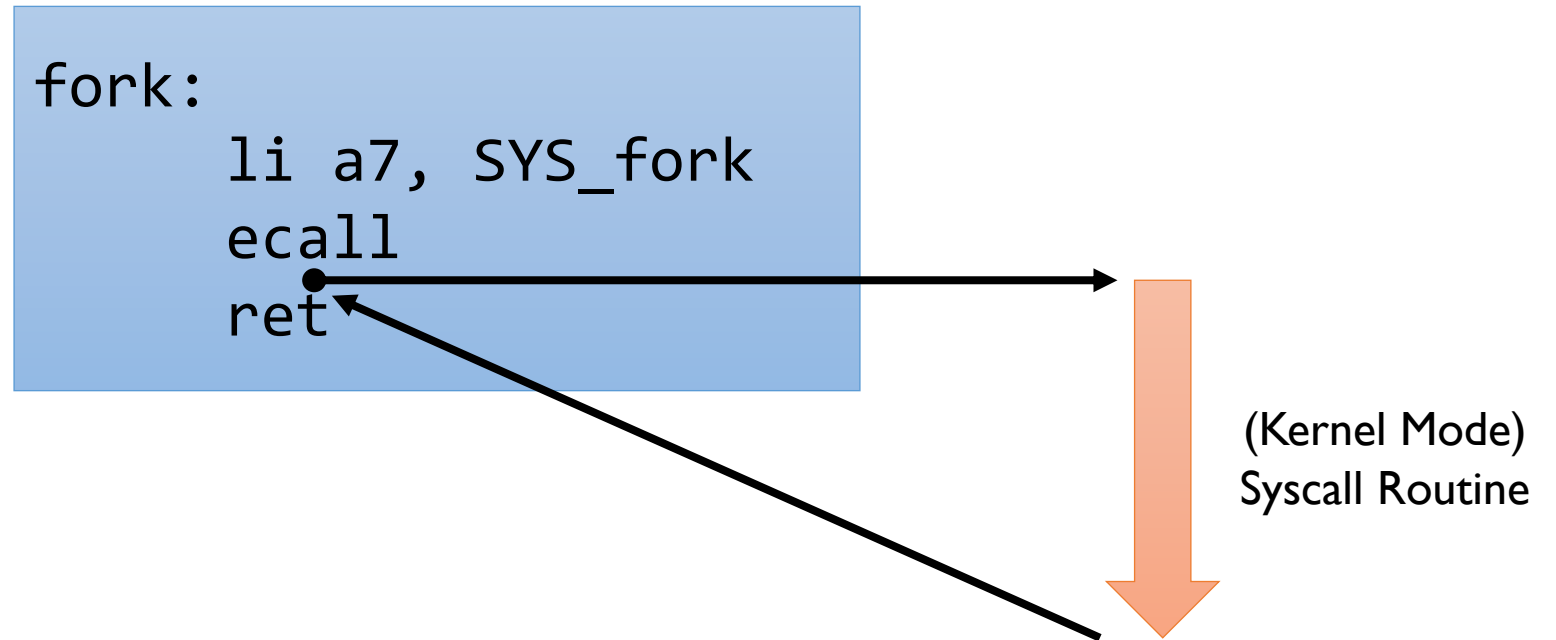
- User applications can access the operating system kernel in a restricted way
- The interfaces that allow user applications to request services from the operating system kernel
- The operating system kernel does the requested task on behalf of user applications

Three RISC-V Privilege Modes

- Machine Mode (M-mode)
 - CPU starts in machine mode
- Supervisor Mode (S-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 (U-mode)
 - User processes run in user mode

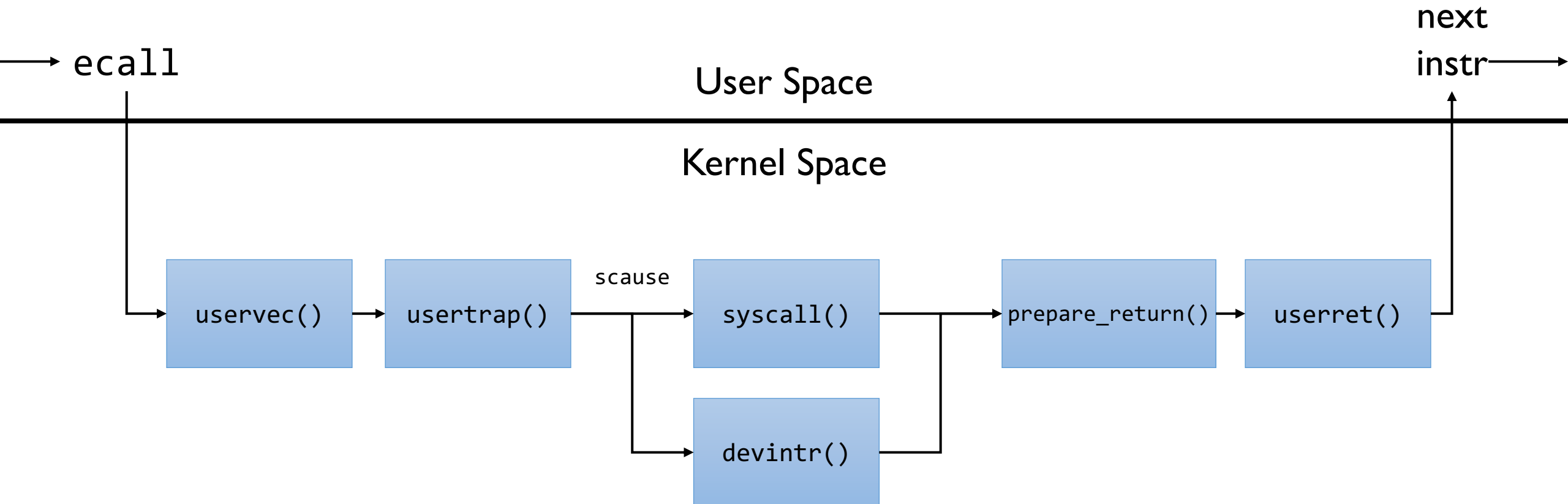
ecall

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



Traps from User Space

- U-mode → S-mode



Some Registers

- `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

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
- 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 `prepare_return()` when it is done
- Jump to `userret()`

prepare_return()

- Install the user trap vector
- Restore user program counter

userret()

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

What Happens on sret

- The RISC-V hart performs all these steps as a single operation
 - Copy the `sepc` into `pc`
 - Start executing at the new `pc`

Physical Memory Protection

■ Physical Memory Protection (PMP)

- A hardware feature that provides fine-grained control over access to memory regions
- It allows the system to define a set of rules governing which memory regions can be accessed by different privilege levels (such as U-mode or S-mode)
- RISC-V supports up to 64 PMP entries, with each PMP entry defined by an 8-bit configuration register (e.g., `pmp0cfg`) and a corresponding 64-bit address register (e.g., `pmpaddr0`)
- Each bit in the PMP configuration register specifies whether the corresponding memory region has permission for read(R), write(W), or instruction execution(X)
- PMP address and configuration registers are only accessible in M-mode

Trap Delegation

- By default, all traps at any privilege level are handled in M-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
- xv6 delegates all interrupts and exceptions to S-mode
 - When the kernel executes the `ecall` instruction in S-mode, control is transferred to the S-mode trap handler (instead of M-mode trap handler)
 - You must make another (nested) system call from S-mode to M-mode to access PMP registers

Project #2

I. Implement the nenter() system call (30 points)

```
int nenter();
```

- It returns the total count of [ENTER] key presses from the console input device since the system booted
- The system call number is already assigned to 23

Project #2

2. Implement the `getpmpaddr()` system call (50 points)

```
void *getpmpaddr(int n);
```

- It returns the 64-bit physical address stored in the PMP address register
- The first (and the only) parameter denotes the index of the PMP address register
 - E.g., 0 for `pmpaddr0`, 1 for `pmpaddr1`, etc
- The system call number is already assigned to 24

Project #2

3. Implement the `getpmpcfg()` system call (20 points)

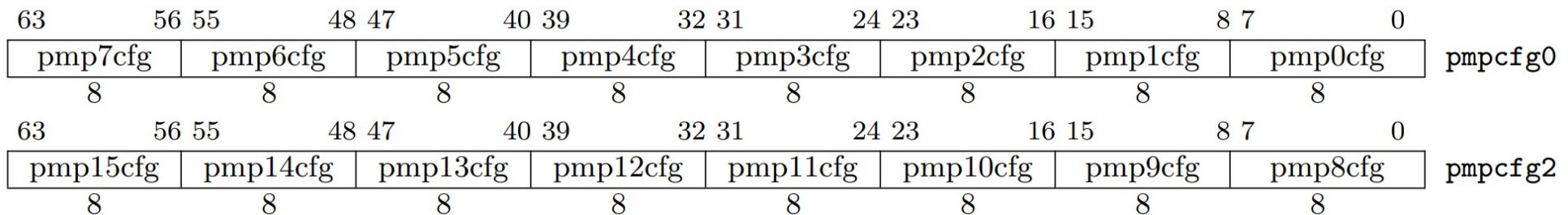
```
int getpmpcfg(int n);
```

- It returns an integer, where the lower 8 bits represent the content of the specified PMP configuration register
- The first (and the only) parameter denotes the index of the PMP configuration register
 - E.g., 0 for `pmp0cfg`, 1 for `pmp1cfg`, etc
- The system call number is already assigned to 25

Project #2

3. Implement the `getpmpcfg()` system call (20 points) (cont'd)

- Since individual 8-bit PMP configuration registers cannot be read directly, you must read the entire 64-bit `pmpcfg0` register



Project #2

■ Tips

- Read Chap. 4.1 of the [xv6 book](#) to understand RISC-V's privileged modes and trap handling mechanism
- Read Chap. 4.2 ~ 4.5 of the [xv6 book](#) to see how traps are handled in xv6
- Read Chap. 5 of the [xv6 book](#) to learn about hardware interrupts
- More detailed information on physical memory protection (PMP) can be found in Chap. 3.7 of the [RISC-V Privileged Architecture manual](#)

Project #2

- You may want to consult:
 - `kernel/console.c`
 - Console-related functions
 - `kernel/kernelvec.S`
 - S-mode trap vectors
 - `kernel/riscv.h`
 - Architecture-dependent codes
 - `kernel/start.c`
 - xv6 kernel boot up code
 - `kernel/syscall.c`
 - General system call handling
 - `kernel/sysproc.c`
 - Several system call implementations
 - `kernel/trap.c`
 - Trap handling
 - And other files if necessary

Project #2

■ Restrictions

- You should use the QEMU version 8.2.0 or higher
- Do not change the predefined system call numbers
- You only need to change the files in the `kernel` directory
- Do not change the `kernel/pmp.c` file

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 nenter and pmptest which can be built from the user/nenter.c and the user/pmptest.c, respectively

Project #2

■ Due

- 11:59 PM, September 27 (Sunday)

■ Submission

- Run `make submit` 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

Using GDB with QEMU

GDB with QEMU

- GDB installation

- Run `sudo apt install gdb-multiarch` (Linux)
- Run `brew install riscv64-elf-gdb` (macOS)

GDB with QEMU

- In the `xv6-riscv-snu` directory, run `make qemu-gdb` to run QEMU
- In another shell, run `gdb-multiarch ./kernel/kernel` (Linux)
- In another shell, run `riscv64-elf-gdb ./kernel/kernel` (macOS)

```
cs1@sys.snu.ac.kr
cs1@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
```

```
cs1@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

- In GDB, enter target remote :<port>
 - You can find TCP port in the QEMU log
 - If you already see output like 0x000000000000001000 in ?? (), skip this step

```
cs1@sys.snu.ac.kr
cs1@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
```

```
cs1@sys.snu.ac.kr
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/cs1/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/cs1/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/cs1/.gdbinit".
To completely disable this security protection add
  set auto-load safe-path /
line to your configuration file "/home/cs1/.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
0x000000000000001000 in ?? ()
(gdb)
```

GDB with QEMU

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

```
cs1@sys.snu.ac.kr
cs1@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)

```
cs1@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/cs1/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/cs1/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/cs1/.gdbinit".
To completely disable this security protection add
  set auto-load safe-path /
line to your configuration file "/home/cs1/.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
- Then the xv6 virtual machine stops immediately

```
cs1@sys.snu.ac.kr
cs1@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)

```
cs1@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/cs1/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/cs1/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/cs1/.gdbinit".
To completely disable this security protection add
    set auto-load safe-path /
line to your configuration file "/home/cs1/.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

```
cs1@sys.snu.ac.kr
cs1@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)

```
cs1@sys.snu.ac.kr
Reading symbols from kernel/kernel...
warning: File "/home/cs1/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/cs1/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/cs1/.gdbinit".
To completely disable this security protection add
    set auto-load safe-path /
line to your configuration file "/home/cs1/.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

```
cs1@sys.snu.ac.kr
cs1@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)

```
cs1@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

```
cs1@sys.snu.ac.kr
cs1@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)

```
cs1@sys.snu.ac.kr
set auto-load safe-path /
line to your configuration file "/home/cs1/.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) █
```

More about GDB

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

Thank you!