Project #3: CPU Hotplugging

Minwook Kim
(ace@snu.ac.kr)
Systems Software & Architecture Lab.
Seoul National University

2026.04.10

Background

CPU Hotplugging

- Dynamically enabling and disabling CPUs (harts) at runtime

- Real-world use cases
  - Power management in servers
  - Elastic scaling in cloud environments
  - Fault isolation and recovery

- In this project: implement system calls to turn harts on/off in xv6

Three RISC-V Privilege Modes (Review)

- Machine Mode
  - Highest privilege; can access all hardware registers
- Supervisor Mode
  - OS kernel runs here
  - Cannot directly access M-mode CSRs (mie, mstatus, …)
- User Mode
  - User processes run here
- This project requires working in both S-mode and M-mode

S-mode → M-mode Traps

- ecall from S-mode triggers a trap into M-mode
  - Handled by the machine trap vector (mtvec)
- Machine Software Interrupt (MSI)
  - Asynchronous — can interrupt a running hart
- Both paths land in the M-mode trap handler

M-mode CSRs

- mcause
  - Trap cause; MSB distinguishes interrupt (1) vs exception (0)
- mepc
  - PC at the time of trap
  - For exceptions: points to the trapping instruction itself
  - For interrupts: points to the next instruction
- mscratch
  - A scratch register for M-mode use (e.g., stack pointer swap)
- mtvec
  - M-mode trap vector address
- mie / mstatus.MIE
  - Per-interrupt enable bits / global M-mode interrupt enable

What Happens on M-mode Trap

- The hart performs all these steps atomically:
  - Store **mstatus.MIE** to **mstatus.MPIE**
  - Disable interrupts: set **mstatus.MIE** to 0
  - Store the current privilege mode into **mstatus.MPP**
  - Copy the pc into **mepc**
  - Set **mcause** to reflect the trap's cause
  - Copy **mtvec** to the pc
  - Switch to M-mode and start executing at the new pc

What happens on mret

- The hart performs all these steps atomically:
  - Restore **mstatus.MIE** from **mstatus.MPIE**
  - Restore the privilege mode from **mstatus.MPP**
  - Set **mstatus.MPIE** to 1
  - Copy **mepc** into pc
  - Start executing at the new pc

Trap Delegation

- By default, all traps are handled in S-mode
  - medeleg and mideleg registers control delegation

- To handle a trap in M-mode, clear the corresponding bit
  - Environment call from S-mode (ecall)
  - Machine software interrupt (MSI)

CLINT and MSI

- A per-hart memory-mapped interrupt controller
- MSIP register: base address 0x2000000 + 4 × hartid (kernel/memlayout.h)
  - Write 1 → trigger MSI on the target hart
  - Write 0 → clear the pending interrupt
- Level-triggered
  - The interrupt stays pending until software clears MSIP
  - Writing 1 again while already set has no additional effect
- When MSIP is pending and mie.MSIE is enabled:
  - The trap is taken into M-mode via mtvec

WFI (Wait For Interrupt)

- wfi instruction: halt the hart until an interrupt arrives
- If an interrupt is already pending, wfi returns immediately
  - This is why other interrupts must be masked before parking
- Used in this project to park an offlined hart
  - Park until an MSI is received from harton

Project #3

Project #3: CPU Hotplugging

- Your task is to implement CPU hotplugging in xv6
- Implement 5 system calls:
  - hartid() — return current hart ID
  - hartmask() — return bitmask of online harts
  - hartpin(mask) — pin calling process to specific hart(s)
  - hartoff(hartid) — turn off a hart
  - harton(hartid) — turn on a hart

20 points

60 points

- Implement the M-mode trap handler (machinevec.S)
  - Handle ecall-from-S and MSI in M-mode

Skeleton Code Overview

- kernel/hotplug.h
  - extern int nharts (actual number of harts, set during boot)
- kernel/hotplug.c
  - Stub functions for all 5 system calls (all return 0)
  - hotplug_init() — called in M-mode during boot (start.c)
- kernel/machinevec.S
  - Empty M-mode trap handler — you fill this in
- kernel/memlayout.h
  - CLINT, CLINT_MSIP(hartid) already defined
- user/hart.c, user/task.c
  - Test programs provided

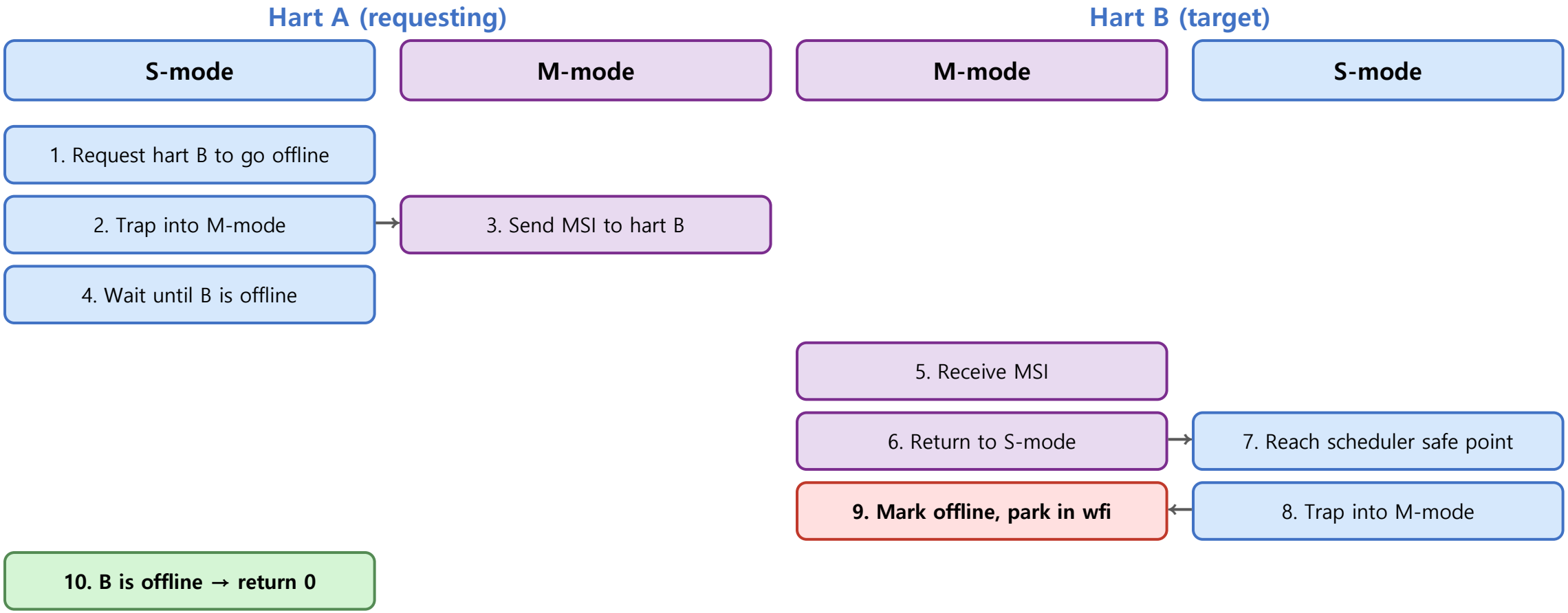
hartpin

- hartpin(mask)
  - Pin the calling process to specific hart(s)
  - mask = 0 → unpin (run on any online hart)
  - mask != 0 → bit i set means hart i is allowed
- The scheduler must respect the pinning
- See user/task.c for example usage:
  - hartpin(1) → pin to hart 0 only (0b0001)
  - hartpin(14) → pin to harts 1,2,3 (0b1110)
- Returns 0 on success
- Returns -1 if:
  - mask does not allow any valid hart
  - mask refers only to non-existent harts
  - no currently online hart is permitted by mask
  - mask contains bits outside the range of supported harts

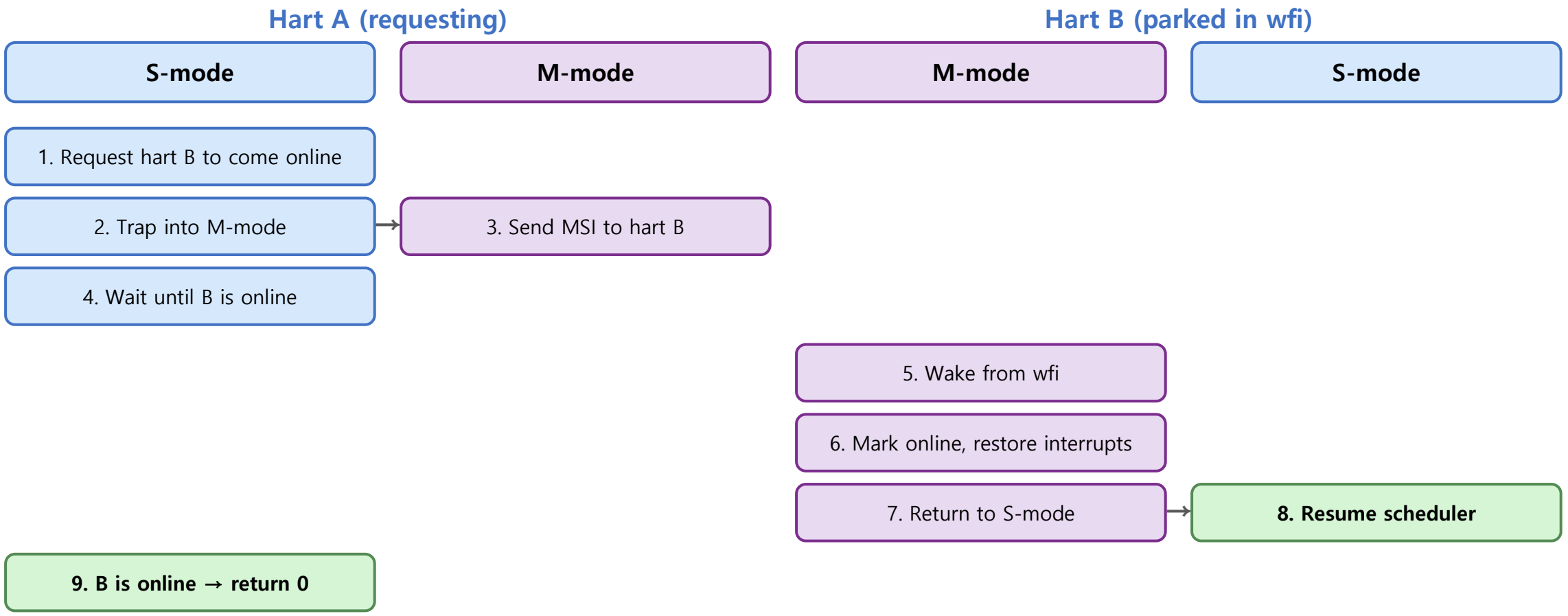
hartoff / harton

- hartoff(hartid)
  - Turn off the target hart
  - The target hart should eventually park itself (e.g., wfi)
  - Return 0 on success
- harton(hartid)
  - Wake the parked hart and bring it back online
  - Return 0 on success
- Error codes (from user/hart.c):
  - -1: hart id out of range
  - -2: cannot turn on/off this hart (e.g., self)
  - -3: already on or off

Conceptual Overview: hartoff() Flow



Conceptual Overview: harton() Flow



Visualizing Scheduling Behavior

- The skeleton provides a logging facility and a visualization tool
  - Kernel built with -DLOG records scheduling events
  - graph.py processes the log and generates a timeline

- Four steps:
  - 1. make qemu-log
  - 2. Run task in the xv6 shell
  - 3. Quit QEMU (Ctrl-a x)
  - 4. Run graph.py

- Output: xv6.log and graph.png

Generating the Scheduling Graph

Step 1: Build and run with logging enabled

```
$ make qemu-log
```

Step 2: Run task in the xv6 shell

```
xv6 kernel is booting

hart 2 starting
hart 1 starting
hart 3 starting
…
init: starting sh
$ task                    <--- type 'task' here
```

Step 3: Wait for task to finish, then quit QEMU

```
$ task
...                       <--- wait ~10 sec
$                         <--- shell prompt returns
                          <--- press Ctrl-a then x
```

Output:

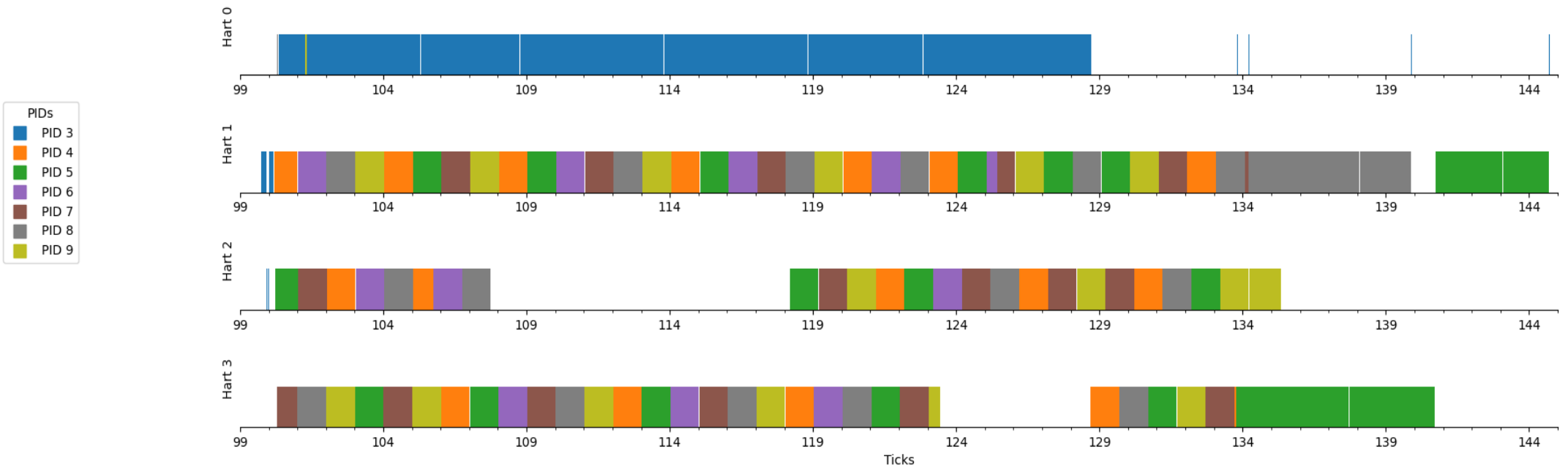
```
QEMU: Terminated
*** The output of xv6 is logged in the 'xv6.log' file.
graph saved in the 'graph.png' file
*** See graph.png file.
```

Log format:

```
11659280 2 starts on 3
11676600 2 ends   on 3
(time)(pid)(action)(hart)
```

You can also generate the graph from an existing log:
$ ./graph.py xv6.log graph.png

Example: graph.png from task.c

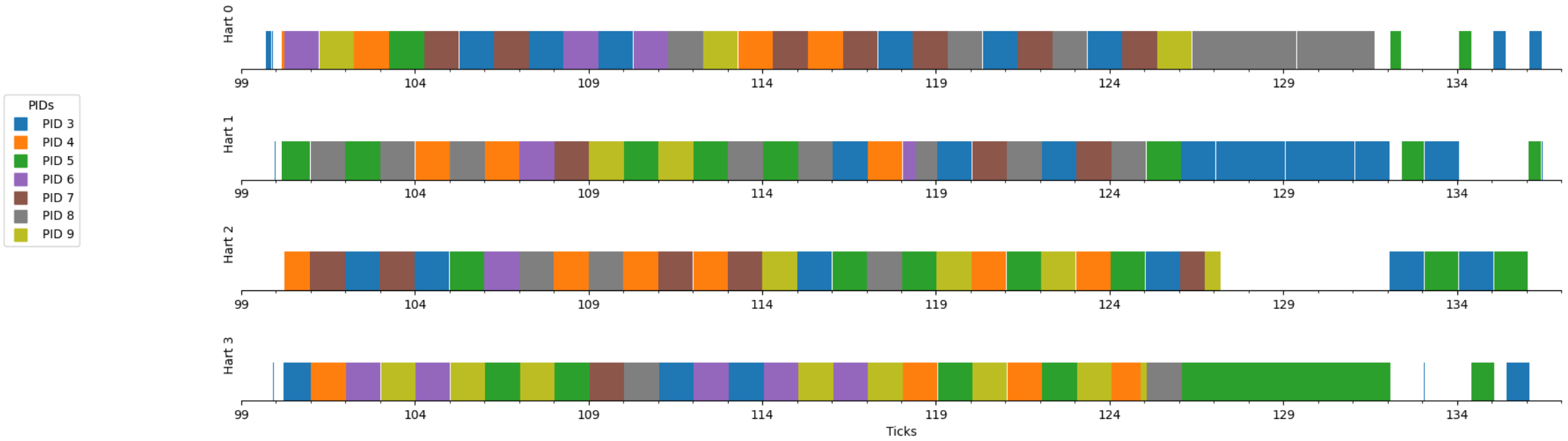


Hart 0: parent (PID 3) pinned via hartpin(1)

Harts 1,2,3: children pinned via hartpin(14)

Gaps = hart offline (hartoff/harton)

Graph Before Implementation



Hints

Hint: Why Not Park Immediately?

- When the target hart receives MSI, it may be holding a spinlock
  - Parking while holding a lock → deadlock
  - The target hart must reach a safe point first

- What is a safe point?
  - No process running on this hart
  - No locks are held

  → The scheduler loop, before scanning for processes

Hint: MSIP Is Level-Triggered

- After handling an MSI, you must clear CLINT_MSIP(hartid)
  - Write 0 to CLINT + 4 * hartid

- If MSIP stays high:
  - mret → interrupt immediately pending → trap again

  → Infinite loop

- Also applies when waking from wfi
  - Clear MSIP before proceeding

Restrictions & Submission

Restriction

- You are allowed to modify only files in the **./kernel** directory :
- Even if you modify other files, those changes will not be committed on t he grading server
- qemu version 8.2.0 or later
  - $ qemu-system-riscv64 --version
- Do not change the system call numbers for hart*

Submission Guidelines

- "make submit" 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 graded

Due Date

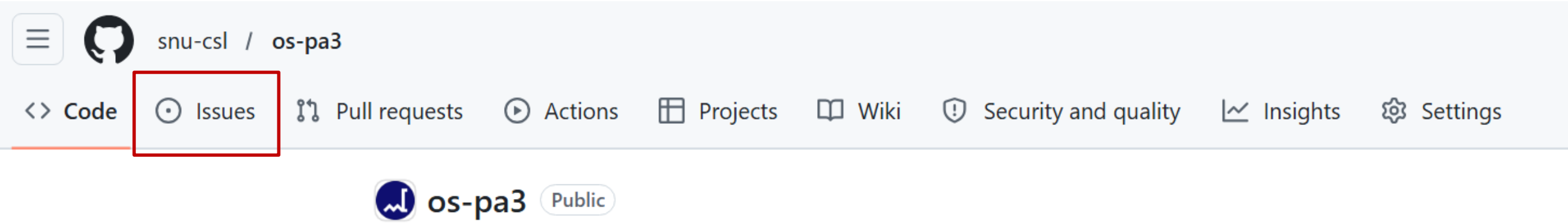
- **Due: 11:59 PM, May 3** (Sunday)
- Only the upload submitted before the deadline will receive full credit.
- 25% of the credit will be deducted for every single day delayed.

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 assignment is announced
  - Once used, slip days cannot be canceled

Q&A

- Please post project-related questions on the *Issues* tab of https://github.com/snu-csl/os-pa3
- Do not upload code



Project #3 Repo

- Skeleton Code
  - Work on the pa3 branch of the xv6-riscv-snu repository:

```
$ git clone https://github.com/snu-csl/xv6-riscv-snu
$ git checkout pa3
```

Thank you!