Project #4: Holy COW!

Sujin Park
(sujinp@snu.ac.kr)
Systems Software &
Architecture Lab.
Seoul National University

2026.05.12

Copy-on-Write (COW)

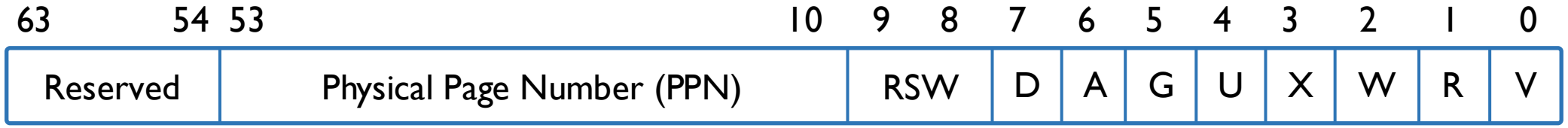
- Virtual memory technique that lets multiple address spaces share the same physical memory until one tries to modify it.

- Instead of eagerly copy all user memory during fork(), the kernel marks the shared pages as read-only and creates a private copy only when a write fault occurs.

- Goal of this project
  - Implement COW in xv6 in two stages

RISC-V Sv39 Paging

- 39-bit virtual address with a three-level page table structure
- **satp** register points to the physical address of the root page table (L2)
- Page-table: 4KiB, 512 PTEs (PTE: 8B,9-bits index)
  - Non-leaf PTE: points to the next-level page table
  - Leaf PTE: points to a physical data page & stores the permission bits



Page Faults

- When access permissions or page table entry validity fails, RISC-V triggers a trap
- **scause** register
  - 12 – Instruction page fault
  - 13 – Load page fault
  - 15 – Store page fault
  - *r_scause()*
- **stval** register contains the faulting virtual address
  - *r_stval()*

Project #4 Overview

- Part 1: ptpages() system call (10 pts)
- Part 2: COW for user address-space pages (30 pts)
- Part 3: COW for leaf (L0) page-table pages (50 pts)
- Part 4: Design Document (10 pts)
- Correctness Requirement: Multi-hart Support without Memory Leaks
- BONUS: Memory-efficiency (up to an additional 10 pts)

Part 1. ptpages() System Call

- System call number 40 in kernel/syscall.h

- int ptpages(int level);
  - Return value
    - The number of user page-table pages allocated at the given level
    - -1 if level is not 0, 1, or 2

- Kernel maintains int ptpages[3] globally
  - Increment when a user page-table page is allocated
  - Decrement when a user page-table page is freed
  - Do not count kernel-only page-table pages

Part 2. COW for User Pages (1)

- Goal: avoid eager page copy during fork()

- When fork(): for each valid, user-accessible, writable page in parent
  - Map the same physical page at the same VA in the child
  - Clear PTE_W in both parent and child PTEs
  - Mark both as COW using a software-defined PTE bit (in RSW)

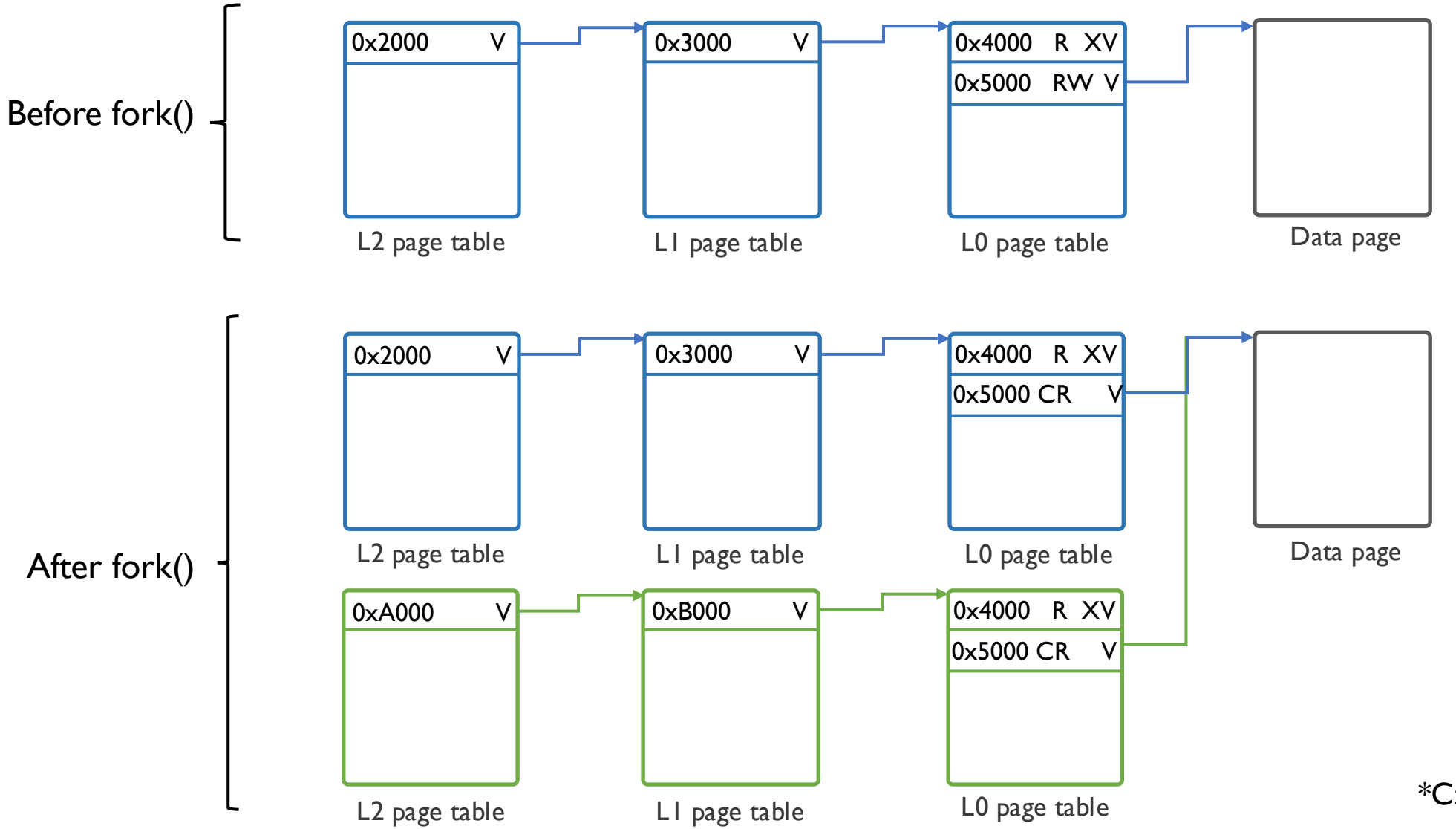
- Pages NOT marked as COW
  - Read-only code pages (already shared, never need write)
  - Trapframe and trampoline pages (special kernel-related mappings)

Part 2. COW for User Pages (2)

- If either process attempts to write to a COW page, the write will cause a store page fault

- Page-fault Handler
  - Recognize that the faulting address refers to a valid COW mapping
  - Allocate a new physical page
  - Copy old page's contents
  - Update only the faulting process's PTE (+ restores PTE_W, clears PTE_COW)
    - Kernel must ensure that a physical page is not freed while it is still shared.

Part 2. COW for User Pages (3)



*C: COW

Part 3. COW for Leaf (L0) Page Tables (1)

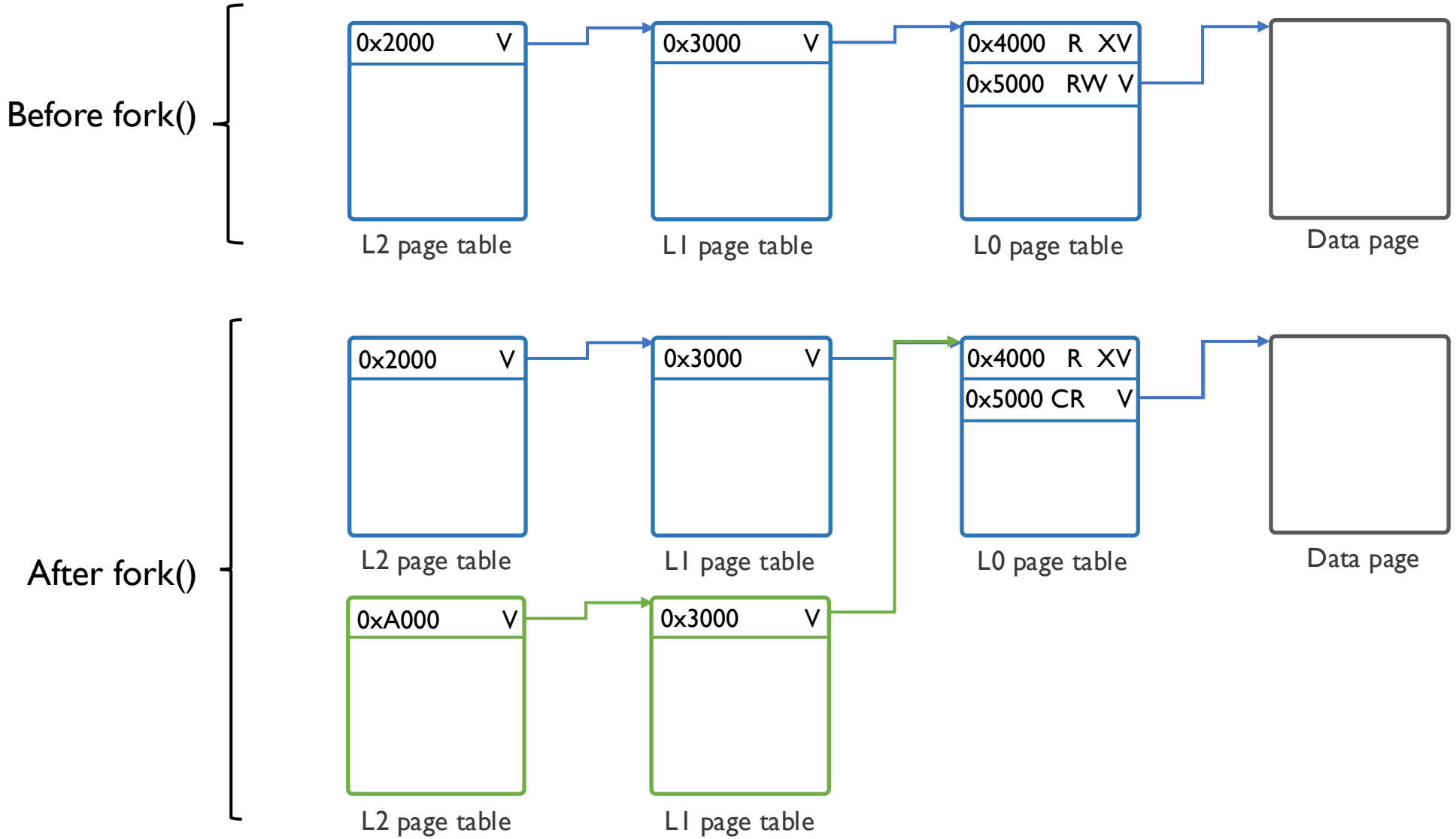
- Goal: reduce additional memory overhead by sharing the L0 page-table
  - Each L0 page contains 512 PTEs, and covers 2MiB of virtual address space
  - Sharing L0 page substantially reduces memory overhead for programs that fork process with large or sparse address space

- After fork(), parent and child may share:
  - The physical data page (Part 2)
  - The leaf page-table themselves (Part 3)

Part 3. COW for Leaf (L0) Page Tables (2)

- Constraint
  - A process must not directly modify a PTE inside an L0 page table that is still shared with another process
  - ⇒ Before modifying a PTE in a shared L0, the kernel must first give the current process its own private L0
- Two distinct forms of sharing (Page-table page vs. Physical memory page)
  - A private L0 does not imply private data pages
- Part 3 must not break Part 2's guarantees
  - Adding L0 sharing must not weaken data-page COW correctness
  - Physical pages must not be freed while they are still referenced by another process
  - Trapframe, trampoline page must still be excluded from COW

Part 3. COW for Leaf (L0) Page Tables (3)



Part 4. Design Document

- Submit a single PDF design document

1. New data structures
   - What you added or modified, and why
2. Algorithm design
   - Reference counting for physical pages and L0 page-table pages
   - fork(), trap handler, page-table sharing
   - Affected system calls / kernel paths and why
   - Corner cases and how you address them
   - Optimizations (time and spaces)
3. Testing and validation
   - Test cases you wrote, corner-case verification
   - Which part took most of your time and why

Correctness Requirement (1)

- Race-free & deadlock- free on multi-hart systems
  - Concurrent page faults, forks, exits, and address-space changes must not corrupt memory or page tables
- No memory leaks
  - All page frames and kernel allocations must be freed
- Avoid Double frees
  - Shared physical page & L0 page table

Correctness Requirement (2)

- Detection mechanism
  - freemem: decremented on kalloc(), incremented on kfree()
    - Provided by the skeleton code
  - ptpages[]: must be restored to pre-run values after a user program returns to sh
  - Press ctrl+p(^p) on the xv6 console to display current values

- Full credit for Part 2 & Part 3
  - Pass ($ usertests –q) on a multi-hart configuration (CPUS > 1 in Makefile)
  - No memory leaks reported by either the freemem counter or the ptpages[] counters

Bonus: Memory-efficiency

- Ranked by the amount of memory used by the kernel for COW-related metadata and page-table management
  - Value of freemem when initial sh starts
  - Minimum freemem observed during the Part 3 stress benchmark

- Awards: up to an additional 10 points
  - Top 10% eligible submissions: +10%
  - Next 10%: +5%

Restrictions

- Compiler flags
  - Part 2 graded with –DPART2; enclose Part 2 code in #ifdef PART2 ... #endif
  - Part 3 graded with –DPART3

- Modify only files in ./kernel
  - other changes are ignored

Tips

- You may modify only the following files
  - kernel/riscv.h | +
  - kernel/kalloc.c | +++++
  - kernel/vm.c | ++++++++++++++++++
  - kernel/proc.c | +
- You can repurpose the RSW field (PTE bits 9-8)
  - Use one bit to mark that a PTE is a COW mapping
- Whenever you modify a valid PTE, call sfence_vma() before returning to user mode

Example: cowtest

- ./user/cowtest.c
- Heap layout: 20MiB aligned to 2MiB boundary
  - Each L0 covers 2MiB → heap spans exactly 10 L0 page tables
- Workflow
  - Parent populates entire 20 MiB heap
  - Parent forks 10 children
  - Each child write one page in each 2 MiB region (10 pages each)
  - Parent verifies its heap is unchanged
  - All children exit

Example: cowtest

```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$
1 sleep  init
2 sleep  sh
freemem=32497
ptpages=[L2:2, L1:4, L0:4]

$ cowtest
cowtest: heap size=20 MiB NL0=10 NCHILD=10
freemem=32482 ptpages=[L2:3 L1:6 L0:6]; PHASE 0: Start
padding 507 pages
aligned heap base=0x0000000000200000 (size=20 MiB)
freemem=27352 ptpages=[L2:3 L1:6 L0:16]; PHASE 1: After parent populates
freemem=27282 ptpages=[L2:13 L1:26 L0:36]; PHASE 2: After parent forks children
freemem=27082 ptpages=[L2:13 L1:26 L0:136]; PHASE 3: After children write
freemem=27082 ptpages=[L2:13 L1:26 L0:136]; PHASE 4: After parent verifies
freemem=27352 ptpages=[L2:3 L1:6 L0:16]; PHASE 5: After all children exit
cowtest: OK
$
1 sleep  init
2 sleep  sh
freemem=32497
ptpages=[L2:2, L1:4, L0:4]
```

- Initial value of freemem can vary depending on your implementation.

cowtest: Phases 0 → 2

- Phase 0 → 1: Parent populates 20 MiB heap
  - 5120 data pages + 10 new L0 page-table pages allocated
  - freemem decreases by 5130
  - ptpages[L0] increase by 10 (L1, L2 unchanged)

- Phase 1 → 2: Parent forks 10 children
  - Heap data pages shared (Part 2)
  - Heap L0 page tables shared (Part 3)
  - Each child still needs: trapframe + L2 + L1 + L0 (for trapframe/trampoline mappings)
  - Each child also touches its stack → 1 COW data page + 1 private L0
  - freemem -70 (7 per child); ptpages L2: +10, L1: +20, L0: +20
    - remain: 10 trapframe pages, 10 COW stack pages

cowtest: Phases 3 → 5

- Phase 2 → 3: Each of 10 children writes 1 page in each of 10 L0 regions
  - Each write triggers two COW
  - 1. Private copy of the L0 page-table page
  - 2. Private copy of the heap data page
  - freemem -200; ptpages[L0]: +100
- Phase 3 → 4: Parent reads its heap to verify
  - Reads do not trigger COW; nothing changes
- Phase 4 → 5: All children exit, parent waits
  - Each child frees: trapframe + private page-table / stack copy / COW pages
  - Counters return to Phase 1 values
    - Confirms no leaks

Skeleton Code

- pa4 branch of xv6-riscv-snu

```
$ git clone https://github.com/snu-csl/xv6-riscv-snu
$ git checkout pa4
```

Submission & Due Date

- Due: 11:59 PM, May 31 (Sunday)

- Submit: make submit generates xv6-pa4-{STUDENTID}.tar.gz
  - Also upload your design document (PDF) to the server
  - Up to 30 submissions; only the version marked FINAL is graded

- Late penalty: 25% off per day

Q&A

- Please upload project related Q&A to https://github.com/snu-csl/os-pa4/issues
- Do not upload code

Thank you