OS Lab – 5

Madhav Bhardwaj (CS23B035)

September 2025

Explanation

- **kernel/riscv.h**

  Defined bit for COW mappings using RSW bit:

  ```
  #define PTE_COW (1L << 8) // uses RSW bit for copy-on-write
  ```

- **kernel/kalloc.c**

  Added a global reference count array indexed by physical page number. Updated kalloc() and kfree() to initialize, increment, and decrement counts. A page returns to the free list only when its refcount reaches zero.

  ```
  int reference_count[PHYSTOP/PGSIZE];

  void* kalloc(void) {
    void *pa = ...; // allocate as usual
    if(pa)
      refcnt[(uint64)pa >> 12] = 1;
    return pa;
  }

  void kfree(void *pa) {
    if(--reference_count[(uint64)pa >> 12] == 0) {
      // actually free the page
    }
  }
  ```

- **kernel/vm.c**

  Mapped the child to the parent's physical pages instead of copying. If a page is writable, clear W and set PTE_COW in both parent and child. If it is read-only, leave it read-only and shared. After mapping the child, increment the page's refcount.

On a COW fault, allocate a fresh page, copy 4KB from the shared page, update the faulting PTE to be writable and clear PTE_COW, then decrement the old page's refcount and flush the TLB.

```
if((pte & PTE_W) != 0){
  // clear write and set COW in parent and child
  pte &= ~PTE_W;
  pte |= PTE_COW;
  // install into child page table with same PA
  ...
  refcnt[pa >> 12]++; // increment reference count
}
```

When copyout() detects PTE_COW, it calls cow_alloc() to split and then performs the copy.

- **kernel/trap.c**

  In usertrap, handled store page fault (scause==15). If stval¡MAXVA and the PTE has PTE_COW, call cow_alloc; otherwise kill the process.

  ```
  if(r_scause() == 15) { // store page fault
    if(cow_alloc(p->pagetable, va) < 0)
      p->killed = 1;
  }
  ```

- **Makefile**

  Added cowtest to UPROGS so it is compiled with user programs.

Logic

I implemented Copy-On-Write (COW) so fork() shares the parent's physical pages with the child and only allocates a private page on the first write. In uvmcopy, if a parent page was writable, I remap both parent and child PTEs to the same PA with write cleared and PTE_COW bit set. A per-frame reference count tracks how many mappings point to each physical page and a frame is actually freed only when the count drops to 0. On a write, the kernel performs a COW split. For user writes (trap with scause=15) or kernel-to-user writes (copyout), I allocate a new page, copy 4KB, update the writer's PTE to writable (clear PTE_COW), and drop one reference from the old page. The code flushes TLBs after permission changes.