Operating Systems Assignment 3 - Easy

Rishit Jakharia 2022CS11621
Aditya Jha 2022CS11102

1 Swapping Mechanism

The page swapping system in XV6 allows the operating system to move pages between physical memory and disk when memory becomes scarce. The implementation consists of several key components:

- **Swap slot management:** Tracks available disk locations for swapped pages
- **Page-out mechanism:** Moves pages from memory to disk
- **Page-in mechanism:** Restores pages from disk when accessed
- **Victim selection:** Determines which pages to swap out
- **Adaptive threshold adjustment:** Uses α and β parameters to optimize swapping behavior

1.1 Swap Page Out Mechanism

Each slot contains information about page permissions and availability, with functions like `get_free_swap_slot()` and `free_swap_slot()` handling allocation and deallocation.

1.2 Page-Out Mechanism

When memory becomes scarce, the system swaps out pages using the following process:

```
int swap_out_page(char *page, int perm) {
    int slot_index = get_free_swap_slot();
    if (slot_index < 0)
        return -1;

    swap_slots[slot_index].page_perm = perm;
    block_no = FSSIZE - SWAP_BLOCKS_TOTAL + slot_index*8;

    // Write page to disk blocks
    for (i = 0; i < 8; i++) {
        bp = bread(1, block_no + i);
        memmove(bp->data, page + i*512, 512);
        bwrite(bp);
        brelse(bp);
    }

    return slot_index;
}
```

This function saves the page's content to disk and returns the slot index where it was stored.

1.3 Page-In Mechanism

When a process attempts to access a swapped-out page, a page fault occurs, triggering the `handle_page_fault` function:

```
int handle_page_fault(uint va) {
    // Get process and validate address
    struct proc *p = myproc();
    va = PGROUNDDOWN(va);

    // Find the page table entry
    pte = walkpgdir(pgdir, (void*)va, 0);

    // Extract swap slot index
    slot_index = ((*pte) >> 12) - 1;

    // Allocate a new physical page
    mem = kalloc();
    if(!mem) {
        // Try to swap out another page if memory is full
        if(swap_page_on_demand() < 0)
            return -1;
        mem = kalloc();
        if(!mem)
            return -1;
    }

    // Read the page from swap
    swap_in_page(mem, slot_index);

    // Map the page back into address space
    mappages(pgdir, (void*)va, PGSIZE, V2P(mem), perm);

    // Free the swap slot
    free_swap_slot(slot_index);
    update_page_no(p, 1);
    return 0;
}
```

This function restores the page from disk to memory and updates the process's page table.

1.4 Adaptive Threshold Management with α and β

The system uses two key parameters for adaptive behavior:

- α **(ALPHA)**: Controls how quickly the number of pages to swap increases
- β **(BETA)**: Controls how quickly the memory threshold decreases

```
void check_memory_threshold(void) {
    int free_pages = count_free_pages();
    if (free_pages <= Th) {
        cprintf("Current Threshold = %d, Swapping %d pages\n", Th, Npg);
        for (int i = 0; i < Npg; i++) {
            struct proc *victim = find_victim_process();
            if (!victim)
                break;
            uint victim_va = find_victim_page(victim);
            // Swap out the selected page
            // ...
```

```
        }

        Th -= (Th * BETA) / 100;
        Npg += (Npg * ALPHA) / 100;
        if (Npg >= LIMIT) {
            Npg = LIMIT;
        }
    }
}
```

This function checks if free memory is below the threshold, swaps out pages, and then adjusts the parameters for next time.

2 Impact of α and β on System Efficiency

2.1 Parameter Relationships

After each swapping operation:

- Threshold decreases: $Th = Th - (Th \times \beta)/100$
- Pages to swap increases: $Npg = Npg + (Npg \times \alpha)/100$

These parameters create a feedback loop that adapts to memory pressure over time.

2.2 Effect of Different Values

2.2.1 Impact of α (Alpha)

- Controls the growth rate of pages swapped per operation
- Higher α means more aggressive scaling of swap size
- With initial $Npg = 4$:
  - $\alpha = 25\%$: After 10 swaps, $Npg \approx 37$ pages
  - $\alpha = 50\%$: After 10 swaps, $Npg \approx 230$ pages

2.2.2 Impact of β (Beta)

- Controls the reduction rate of the threshold
- Higher β means the threshold decreases more rapidly
- With initial $Th = 100$:
  - $\beta = 10\%$: After 10 swaps, $Th \approx 35$ pages
  - $\beta = 25\%$: After 10 swaps, $Th \approx 6$ pages

2.3 Efficiency Considerations

- **High α, Low β**

  System Behavior: Infrequent but large swap operations.

  Efficiency Impact: Good for bursty workloads, poor for steady usage.

  Notes: Memory pressure triggers fewer but more intensive swapping events.

- **Low α, High β**

  System Behavior: Frequent but small swap operations.

  Efficiency Impact: Better I/O distribution, maintains lower memory footprint.

  Notes: System constantly makes small adjustments to memory allocation.

- **High α, High β**

  System Behavior: Aggressive swapping behavior.

  Efficiency Impact: Responsive but may cause thrashing.

  Notes: Quick response to memory pressure but risk of excessive page movement.

- **Low α, Low β**

  System Behavior: Conservative, gradual changes.

  Efficiency Impact: Stable but slower to respond to memory pressure.

  Notes: Prioritizes stability over immediate memory optimization.

The code includes a safety limit (LIMIT = 100) to prevent excessive swapping and potential thrashing.