Assignment - 02.

Ans. 1 Address Translation

→ CPU sees virtual address (VA)

→ MMU splits VA → page number + offset.

→ TLB lookup : if hit → directly gives Physical frame → Physical address (PA)

→ If miss → page table walk → update TLB → form PA.

→ Each process has its own page table to ensure isolation and protection.

Ans 2. Fragmentation

→ Internal : unused space inside allocated block (eg - rounded up page)

→ External : scattered free blocks prevent large requests.

→ fixes (beyond compaction) : Paging, segmentation with paging, buddy allocator, slab / object for small data, huge page, purge dead page allocation

Ans. 3. Paging model and trade off.

→ Memory divided into fixed - size page / frame.

→ Page table + TLB handle mapping

→ Pros : no external fragmentation, flexible allocation.

→ Cons : page table memory overhead, TLB miss add delay, small internal fragmentation inside page.

Ans 4. OS - hardware interaction.

→ Hardware : TLB, MMU, PTEs with R/W/X bits, ASID, page fault mechanism.

→ OS : maintain page table, handle page faults,

manage TLB shootdown.

Example: invalid access → trap → OS loads page → update PTE / TLB → process.

Ans 5. Virtual address width = 16 bits = total virtual address space = 2^{16} bytes.

$2^{16} = 65,536$ bytes.

→ Page size = 1KB = 1024 bytes = 2^{10} bytes.

→ Number of pages = $\frac{2^{16}}{2^{10}} = 2^6 = 64$ pages.

→ Page Table size (each PTE = 2 bytes): 64×2 bytes = 128 bytes.

Answer – 64 pages, page table = 128 bytes.

Ans 6. (stepwise)

① Allocate P1 (212 KB) from the 1000 KB block.

Remaining free = 1000 − 212 = 788 KB.

P1: 212, free → 788 KB.

② Allocate P2 (417 KB) from the first free block (788 KB)

Remaining free = 788 − 417 = 371 KB.

[P1: 212] [P2: 417] [free 371].

③ Allocate P3 (112 KB) from the first free block (371 KB)

Remaining free = 371 − 112 = 259 KB.

[P1: 212] [P2: 417] [P3: 112] [free: 259].

④ Attempt P4 (426 KB): only free block is 259 cannot allocate.

→ After P1, P2, P3 allocated → 259 KB unused; P4 cannot be allocated.

Ans. 7

Reference string : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 (13 refs)

frames = 3.

→ FIFO : 10 faults.

→ LRU : 9 faults.

→ Optimal : 7 faults. (minimum)

Observation → Optimal is best. LRU is close to optimal and better than FIFO. FIFO may suffer Belady's anomaly, but LRU & optimal (stack algorithms) do not.

Ans 8. (a) Overhead.

→ Dirty page : 0.3 × 1000 = 300

→ Disk write : 300 × 10 ms = 3000 ms = 3 s.

→ Memory write : 0.1 ms (negligible)

Extra overhead ≈ 3 seconds.

(b) Optimizations.

→ Prefer clean page as victim.

→ Asynchronous write behind

→ Batch / coalesce writes to reduce latency.

→ Compression before writeback.

→ Use SSD/NVMe for faster writes.

→ Use SSD/NVMe for faster writes.

→ Use SSD/NVMe for faster.

→ Reduce Replacement.

Ans. 9 Autonomous vehicle memory.

(a) Use working set model : keep mission critical page pinned ; replace page from infotainment first.

(b) Strategy : ensure memory pool for real-time tasks, use dynamic pool for others, adopt huge page and prefetching for efficiency