PAGE NO.:
DATE: / /

ASSIGNMENT-2

(1) In Modern systems, each process uses its own logical (virtual) addresses. These are translated into physical addresses in main memory through the Memory Management Unit (MMU) using a page table.

- CPU generated a logical address (virtual address)
- MMU consults the page table to find the corresponding physical frame.
- Physical address is the accessed in RAM.
- This allows multiple processes to coexist safely without interfering with each other.

Illustration

Process A (Virtual Address: 0x1234) → Page Table A
VA 0x1234 → PA 0xA234

Process B (Virtual Address: 0x1234) → Page Table B
VA 0x1234 → PA 0xB678

Physical Memory (RAM)
Frame 0xA234 → Process A's data
Frame 0xB678 → Process B's data

Modern systems use virtual memory where each process generates a logical address. The MMU translates it

into a physical address using page tables, ensuring isolation between processes

(2) Memory layout

| P1 (10) | Free (5) | P2 (20) | Free (10) | P3 (15) |

- External fragmentation : 15 MB free but scattered → can't fit a 12 MB process
- Internal fragmentation : If P4 needs 18 MB but gets 20 MB, 2 MB wasted inside.

Mitigation (beyond compaction).

Modern OS uses :

- Paging → avoids external fragmentation (only last page waste).
- Buddy system / slab allocation → reduces internal fragmentation.

Fragmentation arises from scattered free blocks (external) and oversized allocations (internal).
Modern OS, mitigate using paging, buddy system, or slab allocation, not just compaction.

(3) Paging - based Memory Allocation Model

- Model :
  - Divide physical memory into fixed - size frames (eg. 4KB)
  - Divide process memory into same - sized pages.
  - Use a page table to map virtual pages → physical frames.

PAGE NO.:
DATE: / /

- Trade-off:
  - No external fragmentation (pages can go anywhere)
  - Internal fragmentation possible in the last page.
  - Memory overhead (page tables consume extra space)
  - Speed penalty (extra translation step, mitigated by TLB cache).

(4) OS – Hardware Interaction in Virtual Memory

- Process:
  - CPU generates a virtual address
  - MMU (Memory Management Unit) translates it to a physical address via page tables.
  - TLB (Translation Lookaside Buffer) caches recent translations → speed up access
  - Page Fault: If page not in RAM, OS handles by fetching from disk (swap).
- Key Hardware Structures:
  - MMU → performs translation & enforces protection
  - Page Table Base Register (PTBR) → points to current process's page table.
  - TLB → fast cache of translations.
  - Protection bits (R/W/X in page table

(5) Given
- Virtual address size = 16 bits
- Page size = 1KB = 2^{10} bytes
- Page table entry (PTE) size = 2 bytes.

PAGE NO.:
DATE : / /

a) Number of Virtual Pages

- Virtual address space = 2^{16} = 65,536 bytes (64KB).
- Page size = 2^{10} = 1024 bytes.
- Number of virtual pages = $\frac{\text{Virtual Address Space}}{\text{Page size}} = \frac{2^{16}}{2^{10}} = 2^6 = 64$

∴ 64 virtual pages.

b) Pages Table Size

- Each virtual page needs 1 entry
- Total entries = 64
- Size = 64 × 2 bytes = 128 bytes.

∴ Pages table size = 128 bytes

(c) Allocation (all 3 methods):

- P1 → 212 KB, P2 → 417, P3 → 112 KB allocated.
- P4 → cannot fit (largest free block = 259 KB < 426 KB)

~~Used~~ Unused memory:

- First - fit = 259 KB
- Best - fit = 259 KB
- Worst - fit = 259 KB

All three give same utilization here (741 KB used, 259 KB free) since processes arrive in order and split the single block identity.

PAGE NO.:
DATE: / /

(7) Page - replacement - (reference string = 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 frames).

a & b — emulation results (summary):

- FIFO : page faults = 10
- Optimal : page faults = 7
- LRU : page faults = 9.

(Short justification : Optimal by definition produces the minimum fault (7). FIFO replaces the oldest page and here causes 10 faults; LRU (replaces least-recently used) gives 9 faults).

c → comment (Belady's Anomaly):

- Optimal and LRU are stack algorithms (they do not suffer Belady's anomaly) and tend to perform better.
- FIFO is not a stack algorithm and can exhibit Belady's anomaly (faults can increase when frames increase). In this case FIFO performs worse than Optimal and LRU, illustrating that naive replacement (FIFO) may give suboptimal behaviour — exactly the kind of effect Belady described.

(8) Dirty - page overhead and optimization

Given : disk write = 10 ms per page, memory write = 100 ns, 30% of replaced pages are dirty, replacing 1000 pages.

a — extra time due to dirty pages:

- Number dirty = 0.30 × 1000 = 300 pages.
- Time for dirty writes = 300 × 10 ms = 3000 ms = 3 seconds

[for scale: 3 s = 3,000,000,000 ns, which equals 30,000,000 memory writes at 100 ns each — shows disk cost dominates].

b — Optimization to reduce overhead:

- use asynchronous write-behind [page cleaner / write-back daemon] + prefer evicting clean pages.
- A background page-cleaner writes dirty pages to disk ahead of eviction so the page fault handler rarely blocks on disk I/O.
- Combine with an eviction policy that prefers clean pages (eg. enhanced CLOCK / second-chance that checks dirty bit).
- Result: fewer synchronous disk writes during replacement → much lower page fault latency.

(9) a) Keeping mission-critical task from thrashing.

- Working-set tracking: OS measures each task's working set. The scheduler / VM subsystem pins or reserves the estimated working set for critical tasks so they keep their pages in RAM and don't suffer page faults.

PAGE NO.:
DATE: / /

- Priority-aware replacement: Use a WSClock / enhanced-clock replacement that combines working-set age checks with process priority: when reclaiming, prefer to evict pages belonging to low-priority processes (enforcement) and only reclaim a critical task's pages if its working set has genuinely cooled.
  Also prefer evicting clean pages and background-clean dirty pages to avoid I/O stalls.

- Admission & prefetching: Refuse to admit extra high-memory best-effort tasks when critical tasks's working sets cannot be satisfied; prefetch pages for critical tasks (eg. model weights) at startup to avoid runtime faults.

b) Recommended memory allocation strategy & justification.

- Hybrid reservation + dynamic pool:

1. Reserved real-time pool: statically / periodically reserve a fraction of RAM for mission-critical process (mlock / rt cgroup + preallocated pools or huge pages).

2. Shared dynamic pool: remaining memory is managed by the normal VM with priority-aware reclaim for best-effort tasks.

a Supporting techniques: use huge pages for large model data (reduce TLB pressure), slab / pool allocators for predictable kernel allocations, an async page-cleaner daemon, and cgroup

memory QoS to throttle noncritical workloads.

- Why this balances both: the reserved pool guarantees real-time responsiveness and predictable latency for safety-critical modules, the dynamic pool maximizes overall utilization by allowing noncritical services to expand / contract and absorb reclaim / IO cost. Combining priority-aware replacement and prefetch / cleaning minimizes blocking I/O and keeps system efficient.