UNIT IV · LESSON 04

Memory Management & Virtual Memory

binding · contiguous allocation · fragmentation · paging · TLB · segmentation · demand paging · page replacement · thrashing

30
questions in Unit IV
OS · Networks · Cloud

WHEN ADDRESS BINDING

Time	Address type	Note
Compile time	absolute code	must know location in advance
Load time	relocatable code	bind when loaded
Execution time	logical ≠ physical	needs hardware (MMU); most OSes

- **Logical (virtual) address** generated by CPU; **physical** address seen by memory unit
- **MMU** maps logical → physical (relocation register: physical = logical + base)
- **Dynamic loading** (routine loaded when called), **dynamic linking** (shared libraries, stubs)
- **Swapping**: move whole process to backing store and back

PARTITIONS CONTIGUOUS ALLOCATION

- **Fixed partitions**: internal fragmentation, limited multiprogramming
- **Variable partitions**: holes; external fragmentation

Strategy	Choose	Note
First fit	first hole big enough	fast
Best fit	smallest adequate hole	leaves tiny holes
Worst fit	largest hole	leaves big holes
Next fit	first fit from last position	

- First fit & best fit beat worst fit in speed and storage use
- **50-percent rule**: with first fit, N blocks allocated → 0.5 N lost to fragmentation (⅓ of memory)
- Protection: **base + limit** registers

HOLES 100, 500, 200, 300, 600 KB FIT EXAMPLE

Request	First fit	Best fit	Worst fit
212 KB	500 (→288)	300 (→88)	600 (→388)
417 KB	600 (→183)	500 (→83)	500 (→83)
112 KB	288 (→176)	200 (→88)	388 (→276)

HOLES 100, 500, 200, 300, 600 KB FIT EXAMPLE (CONTD.)

Request	First fit	Best fit	Worst fit
426 KB	**waits**	600 (→174)	**waits**

- Only **best fit** places all four here

WASTE FRAGMENTATION

Internal	External
unused space inside an allocated block	free memory split into **non-contiguous** holes
fixed partitions, paging (last page)	variable partitions, segmentation
reduce with smaller units	fix by **compaction** or paging

- Average internal fragmentation in paging = **½ page per process**

WORKED PAGING NUMBERS

- 32-bit logical address, 4 KB pages → offset 12 bits, page number 20 bits → **2^{20} entries**
- PTE 4 B → page table = 4 MB per process → needs **multilevel** paging
- Physical memory 64 MB with 4 KB frames → 2^{14} frames → frame number 14 bits
- Logical address 13 000, page 4096 → page 3, offset 13 000 − 12 288 = **712**

LARGE SPACES PAGE-TABLE STRUCTURES

- **Hierarchical** (two-level: outer page table → inner page table; 64-bit uses 3–4 levels)
- **Hashed** page table (virtual page number hashed, chains)

NON-CONTIGUOUS PAGING

- Physical memory → **frames**; logical memory → **pages** of the same size (power of 2)
- Logical address = **page number p + offset d**
- Logical space 2^m, page size 2^n → p = **m − n** bits, d = n bits
- Physical address = frame number × page size + offset
- **Page table** per process (in memory, **PTBR**, PTLR); entry has frame no., valid/invalid, protection, dirty, reference bits
- No external fragmentation; some internal fragmentation

FAST LOOKUP TLB

- **Translation look-aside buffer**: associative cache of page-table entries
- **EAT = $h(t_{TLB} + t_m) + (1 - h)(t_{TLB} + 2t_m)$** (single-level table)
- TLB 20 ns, memory 100 ns, h = 80 % → 0.8×120 + 0.2×220 = **140 ns**
- h = 98 % → 0.98×120 + 0.02×220 = **122 ns**
- **ASIDs** avoid flushing the TLB on a context switch

LARGE SPACES PAGE-TABLE STRUCTURES (CONTD.)

- **Inverted** page table: one entry per **frame** (pid, page) — saves memory, slow search (use hash)
- Shared pages: **reentrant** code shared between processes

DEMAND PAGING VIRTUAL MEMORY

- Execute processes **not completely in memory**; logical space ≫ physical
- **Demand paging**: load page only when referenced (lazy swapper / pager)
- Reference to invalid page → **page fault** → trap → find free frame → read page from disk → update table → restart instruction
- **EAT = (1 − p)·ma + p·page-fault time**
- ma = 200 ns, fault service 8 ms, p = 1/1000 → EAT ≈ **8.2 µs** (40× slower)
- For < 10 % slowdown → p < 1/399 990 (about 1 fault in 400 000 accesses)
- **Copy-on-write** for fork; **pure demand paging** starts with zero pages

ALGORITHMS PAGE REPLACEMENT (CONTD.)

- **Stack algorithms** (OPT, LRU) never show Belady's anomaly
- **Dirty (modify) bit**: write back only if modified

USER VIEW SEGMENTATION

- Program = segments (main, functions, stack, symbol table, arrays) of **varying** length
- Address = ⟨segment number s, offset d⟩; segment table: **base & limit**
- d < limit else **trap** (addressing error); physical = base + d
- External fragmentation; supports sharing & protection per segment
- **Segmented paging** (Intel x86): segments divided into pages
- Seg 2: base 4300, limit 400 → ⟨2, 53⟩ → **4353**; ⟨2, 500⟩ → trap

ALGORITHMS PAGE REPLACEMENT

Algorithm	Replace	Note
FIFO	oldest page	**Belady's anomaly** possible
Optimal (OPT, MIN)	page used **farthest in future**	lowest faults; needs future → benchmark
LRU	least recently used	stack algorithm, no Belady; counters/stack
LRU approximations	reference bit, **second chance (clock)**, enhanced (ref, dirty)	
LFU / MFU	counting based	rarely used

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 (3 FRAMES) REPLACEMENT TRACE

Algorithm	Page faults
FIFO	**15**
Optimal	**9**
LRU	**12**

Belady's anomaly with FIFO: reference string 1 2 3 4 1 2 5 1 2 3 4 5 → **9 faults with 3 frames, 10 faults with 4 frames**.

ALLOCATION FRAMES & THRASHING

- Allocation: equal, **proportional** ($a_i = s_i/S \times m$), priority-based; global vs local replacement
- **Thrashing**: more time paging than executing; CPU utilisation collapses as degree of multiprogramming rises
- Cause: Σ working-set sizes > available frames
- **Working-set model** (window Δ): WSS_i; if $D = \Sigma\ WSS_i > m$ → suspend a process
- **Page-fault-frequency** scheme: upper/lower bounds on fault rate
- Locality model explains why demand paging works

TRAP EXAM TRAPS

- Paging has internal (not external) fragmentation; segmentation has external
- LRU and OPT never suffer Belady's anomaly; FIFO can
- Two memory accesses per reference without TLB (table + data)
- Offset bits depend only on page size, not on memory size
- Page size = frame size, always

Q → A RAPID RECALL

Maps logical to physical addresses	MMU
Waste inside a partition	internal fragmentation
Remedy for external fragmentation	compaction / paging
Page number bits, 32-bit address, 1 KB page	22
Cache of page-table entries	TLB
Table with one entry per physical frame	inverted page table
Segment table holds	base and limit
FIFO anomaly	Belady's anomaly
Lowest fault rate (theoretical)	optimal (OPT)
Excessive paging	thrashing
Fault rate control scheme	page-fault frequency