MODULE 4

Memory Management

Comprehensive Study Material

Course Code: DCSE250042 | B.Sc. Computer Science (Semester 3)

4.1 Memory Management Fundamentals

4.1.1 What is Memory Management?

Memory Management is the functionality of an operating system that handles the allocation, tracking, and deallocation of primary memory (RAM) to processes.

Why is Memory Management Needed?

- Multiple processes need to reside in memory simultaneously
- Each process must have its own protected memory space
- Memory is limited – must be used efficiently
- Programs can be larger than physical memory
- Fast access to memory is critical for performance

4.1.2 Memory Hierarchy

Level	Size	Speed	Cost/GB	Managed By
Registers	Bytes	0.3 ns	Very High	Compiler
Cache (L1–L3)	MB	1–10 ns	High	Hardware
RAM	GB–TB	50–100 ns	Medium	OS
SSD	TB	0.1 ms	Low	OS/User
HDD	TB–PB	10 ms	Very Low	OS/User

4.1.3 Roles of the Memory Management Unit (MMU)

The MMU is hardware that translates virtual (logical) addresses to physical addresses.

1. Address translation (logical → physical)

2. Memory protection (prevent process from accessing another's memory)
3. Cache control
4. Memory mapping

4.1.4 Base & Limit Registers

The Relocation (Base) Register stores the starting physical address of a process. The Limit Register stores its size.

```
Physical Address  =  Logical Address + Base Register
Valid check       :  Logical Address < Limit Register

Example:  Base = 1000, Limit = 500
  Logical = 250  →  Physical = 1250  (250 < 500 ✓)
  Logical = 600  →  Trap! (600 ≥ 500 – memory violation)
```

4.2 Logical vs Physical Address Space

4.2.1 Definitions

Term	Definition	Who Generates?	Example
Logical Address	Address generated by CPU (virtual)	CPU / Program	0x0045A3F2
Physical Address	Actual address in RAM	MMU	0x00F2B400
Logical Address Space	All logical addresses a program can generate	Programmer view	0 to 2^32-1 (4 GB)
Physical Address Space	All physical addresses in RAM	Hardware	Depends on RAM size

4.2.2 Types of Relocation

Type	Timing	Method	Advantages	Disadvantages
Static Relocation	Load time	Modify addresses before execution	No runtime overhead	Cannot move after loading
Dynamic Relocation	Run time	MMU with base/limit registers	Flexible, supports swapping	Hardware overhead

4.2.3 Swapping

Swapping moves entire processes between main memory and disk to free up RAM for other processes.

```
Swap time = Seek time + Rotational Latency + Transfer time

Example: 10 MB process at 50 MB/s
  Transfer = 10 MB / 50 MB/s = 200 ms
  Total    = 10 ms + 5 ms + 200 ms = 215 ms
```

4.3 Memory Allocation Strategies

4.3.1 Types of Fragmentation

Type	Definition	Example	Solution
Internal Fragmentation	Wasted space INSIDE an allocated block	Partition 8 KB, process 5 KB → 3 KB wasted	Smaller partitions, paging
External Fragmentation	Wasted space BETWEEN allocated blocks	Holes of 2 KB, 1 KB, 4 KB can't fit a 5 KB process	Compaction, paging

4.3.2 Fixed Partitioning

Memory is divided into fixed-size partitions at system boot.

- Advantage: Simple to implement; no external fragmentation
- Disadvantage: Internal fragmentation; maximum process size limited; inefficient for varying sizes

4.3.3 Dynamic (Variable) Partitioning

Partitions are created dynamically to match process size. Produces external fragmentation over time.

Compaction can merge scattered free holes into one large block, but is expensive (requires moving processes).

4.3.4 Dynamic Partitioning Algorithms

Algorithm	Rule	Pros	Cons
First-Fit	First hole large enough	Fast, simple	Can create many small holes at beginning
Best-Fit	Smallest hole large enough	Leaves largest possible hole	Creates tiny useless holes; slow search
Worst-Fit	Largest hole	Leaves medium holes	May leave no large hole for big processes

Algorithm	Rule	Pros	Cons
Next-Fit	Next hole after last allocation	Spreads allocations evenly	May be worse than First-Fit

Empirical result: First-Fit is usually the best choice.

4.4 Paging

4.4.1 Basic Concept

Paging divides physical memory into fixed-size blocks called frames and logical memory into same-size blocks called pages. Pages can be mapped to any free frame – no need for contiguous physical memory.

4.4.2 Address Translation

```
Logical Address  =  Page Number (p)  |  Offset (d)

Physical Address =  Frame Number (f)  |  Offset (d)
  where Frame Number = Page Table[p]

Example: Page size = 4 KB, Logical addr = 0x8A3F
  Page  = 0x8A3F / 4096 = 8
  Offset = 0x8A3F % 4096 = 0xA3F = 2623
  If Page Table[8] = Frame 12:
  Physical = 12 × 4096 + 2623 = 51775 (0xCA3F)
```

4.4.3 Translation Lookaside Buffer (TLB)

The TLB is a small, fast hardware cache for page table entries. It avoids the need for a second memory access on every instruction.

```
EAT = HR × (TLB_time + MA) + (1 - HR) × (TLB_time + 2 × MA)

Where: HR = Hit Ratio, MA = Memory Access Time

Example: TLB = 1 ns, MA = 100 ns, HR = 98%
EAT = 0.98 × 101 + 0.02 × 201
    = 98.98 + 4.02 = 103 ns   (vs 200 ns without TLB)
```

4.4.4 Page Table Implementation Techniques

Hierarchical (Multi-Level) Paging

Pages the page table itself to avoid a huge single-level table. A 32-bit address with 4 KB pages would otherwise need 1 million entries (4 MB per process!).

Inverted Page Table

One entry per physical frame (not per logical page). Dramatically smaller table, but requires hashing to find the frame for a given (PID, page) pair.

4.4.5 Paging: Advantages & Disadvantages

Advantages	Disadvantages
No external fragmentation	Internal fragmentation (last page may not be full)
Processes can exceed physical memory (with virtual memory)	Page table overhead memory
Easy to swap individual pages	Two memory accesses per reference (mitigated by TLB)
Sharing possible via shared pages	Page table can be very large for 64-bit spaces

4.5 Segmentation

4.5.1 Basic Concept

Segmentation divides memory into logical units (code, data, stack, etc.) that match the program's structure. Segments have variable sizes defined by their content.

4.5.2 Address Translation

```
Physical Address = Base[segment] + Offset
Valid check      : Offset < Limit[segment]

Example: Segment 1 (Data), Base=3000, Limit=400
  Logical (1, 150) → Physical = 3000 + 150 = 3150 ✓
  Logical (0,  900) → Fault! (900 ≥ 800 = Limit[0])
```

4.5.3 Paging vs Segmentation Comparison

Feature	Paging	Segmentation
Division basis	Fixed size (mechanical)	Logical units (programmer view)
Block size	Fixed (page = frame)	Variable (segment)
Internal fragmentation	Yes (last page)	Small (if any)
External fragmentation	No	Yes
Sharing	Difficult	Easy (share full segments)
Protection	Per page	Per segment (intuitive)
Programmer visibility	Invisible	Visible

4.6 Virtual Memory

4.6.1 Concept

Virtual Memory allows processes to be larger than physical RAM by keeping only needed pages in memory and storing the rest on disk (backing store).

4.6.2 Benefits

Benefit	Explanation
Processes larger than RAM	A 16 GB program can run on 8 GB RAM
Higher multiprogramming	More processes fit because each uses less RAM
Faster process creation	Only needed pages loaded initially
Less I/O for swapping	Swap only pages, not whole processes
Code sharing	Shared libraries loaded into memory once

4.6.3 Page Table Valid/Invalid Bit

Each page table entry has a valid bit. 1 = page is in RAM; 0 = page is on disk (triggers page fault on access).

4.6.4 Copy-on-Write (COW)

After fork(), parent and child share pages marked read-only. A copy is made only when either process writes to a shared page. This makes fork() very fast and conserves memory.

4.7 Demand Paging

4.7.1 Concept

Demand Paging loads pages into memory only when they are actually referenced, not when the process starts.

4.7.2 Page Fault Handling Steps

5. Trap to OS – save process state
6. Verify the reference is valid
7. Find a free frame (or evict one via page replacement)
8. Read the required page from disk
9. Update page table (set valid = 1, record frame number)
10. Restart the faulting instruction

4.7.3 Effective Access Time (EAT) with Demand Paging

```
EAT = (1 - p) × MA + p × PFT

Where: p = page-fault rate, MA = memory access time, PFT = page-fault
service time

Typical values: MA = 200 ns, PFT = 8 ms = 8,000,000 ns

p = 0        → EAT = 200 ns
p = 0.0001   → EAT ≈ 1,000 ns  (5× slower)
p = 0.001    → EAT ≈ 8,200 ns  (41× slower)
p = 0.01     → EAT ≈ 80,198 ns  (401× slower)

Conclusion: Page-fault rate must be < 0.0001 for acceptable performance!
```

4.8 Page Replacement Algorithms

4.8.1 Overview

When a page fault occurs and no free frames exist, the OS must choose a victim page to evict (writing it to disk if modified – the dirty bit).

4.8.2 FIFO – First-In, First-Out

Replaces the page that has been in memory the longest.

Reference string: 1,2,3,4,1,2,5,1,2,3,4,5 | 3 frames → 9 page faults

⚠️ **Belady's Anomaly: Adding more frames can INCREASE page faults with FIFO.**

4.8.3 Optimal (OPT / MIN)

Replaces the page that will not be used for the longest time in the future. This gives the minimum possible page faults and is used as a benchmark.

Same reference string: 8 page faults (cannot be improved upon).

Cannot be implemented in practice (requires future knowledge).

4.8.4 LRU – Least Recently Used

Replaces the page that has not been used for the longest time (looks backward, not forward).

Same reference string: 10 page faults. Does NOT suffer from Belady's Anomaly.

4.8.5 Second-Chance (Clock) Algorithm

A circular FIFO list where each page has a reference bit. When evicting, if R=1 give a second chance (clear to 0, advance clock hand); if R=0, evict immediately.

4.8.6 Enhanced Second-Chance (Reference + Dirty Bit)

(R, M)	Meaning	Priority for Replacement
(0, 0)	Not used recently, not modified	BEST – no write needed
(0, 1)	Not used recently, modified	Good – need one write
(1, 0)	Used recently, not modified	Not preferred
(1, 1)	Used recently, modified	WORST – keep if possible

4.8.7 Algorithm Comparison

Algorithm	Page Faults*	Implementable?	Belady's Anomaly?
FIFO	9	Yes – very easy	Yes
Optimal	8	No (benchmark only)	No
LRU	10	With hardware help	No
Second-Chance	~9–10	Yes – easy	No

Algorithm	Page Faults*	Implementable?	Belady's Anomaly?
Enhanced 2nd Chance	~9	Yes – moderate	No

* For reference string: 1,2,3,4,1,2,5,1,2,3,4,5 with 3 frames.

4.8.8 Frame Allocation

Replacement Type	Description	Effect
Local Replacement	Victim chosen from same process only	Process's fault rate independent of others
Global Replacement	Victim from any process	One process can affect others; dominant in modern OS

4.9 Thrashing

4.9.1 Definition

Thrashing occurs when a process (or the OS) spends more time swapping pages than executing useful instructions. CPU utilisation collapses despite high paging I/O activity.

4.9.2 Cause

Too many processes → each gets too few frames → high page-fault rate → high I/O → low CPU utilisation → OS adds more processes → cycle worsens.

4.9.3 Locality of Reference

Type	Description	Example
Temporal Locality	Recently used addresses are likely to be used again soon	Loop variable accessed repeatedly
Spatial Locality	Nearby addresses are likely to be accessed soon	Sequential array traversal

4.9.4 Solutions to Thrashing

Solution	Method	Effectiveness
Working Set Model	Keep each process's working set in memory	High
Page Fault Frequency (PFF)	Allocate more frames if fault rate > upper bound; fewer if < lower	High

Solution	Method	Effectiveness
Process Suspension	Suspend lowest-priority process; give its frames to others	Immediate
Better Page Replacement	Use LRU or working-set-aware algorithm	Moderate

```
Working Set W(t, Δ) = set of distinct pages referenced in last Δ time units

If Σ |WS_i| > Available Frames → suspend one or more processes
```

4.10 Numerical Problems with Solutions

Problem 1 – Address Translation (Paging)

Q: Page size = 4 KB, 32-bit logical address. Process accesses 0x8A3F. Page Table[8] = Frame 12. Find the physical address.

```
Page size = 4096 bytes = 2^12  →  12 offset bits
0x8A3F  = 8 × 4096 + 0xA3F  =  8 × 4096 + 2623
Page number = 8,  Offset = 2623
Frame 12: Physical = 12 × 4096 + 2623 = 49152 + 2623 = 51775 = 0xCA3F
```

Problem 2 – EAT with TLB

Q: TLB = 2 ns, Memory = 100 ns, Hit ratio = 95%. Find EAT.

```
EAT = 0.95 × (2 + 100) + 0.05 × (2 + 200)
    = 0.95 × 102 + 0.05 × 202
    = 96.9 + 10.1 = 107 ns
(Without TLB: 200 ns; with TLB: 107 ns ≈ 1.87× faster)
```

Problem 3 – FIFO Page Replacement

Q: Reference string 1,2,3,4,1,2,5,1,2,3,4,5 | 3 frames. Count page faults using FIFO.

Ref	Action	Frame 1	Frame 2	Frame 3	Fault?
1	Load 1	1	–	–	Yes
2	Load 2	1	2	–	Yes
3	Load 3	1	2	3	Yes
4	Evict 1, load 4	4	2	3	Yes
1	Evict 2, load 1	4	1	3	Yes

Ref	Action	Frame 1	Frame 2	Frame 3	Fault?
2	Evict 3, load 2	4	1	2	Yes
5	Evict 4, load 5	5	1	2	Yes
1	1 in memory	5	1	2	No
2	2 in memory	5	1	2	No
3	Evict 1, load 3	5	3	2	Yes
4	Evict 2, load 4	5	3	4	Yes
5	5 in memory	5	3	4	No

Total page faults = 9

Problem 4 – LRU Page Replacement

Q: Same reference string, 3 frames, LRU. Count page faults.

Ref	LRU Evicts	Frames (MRU→LRU)	Fault?
1	–	[1]	Yes
2	–	[2,1]	Yes
3	–	[3,2,1]	Yes
4	1 (LRU)	[4,3,2]	Yes
1	2 (LRU)	[1,4,3]	Yes
2	3 (LRU)	[2,1,4]	Yes
5	4 (LRU)	[5,2,1]	Yes
1	–	[1,5,2]	No
2	–	[2,1,5]	No
3	5 (LRU)	[3,2,1]	Yes
4	1 (LRU)	[4,3,2]	Yes
5	2 (LRU)	[5,4,3]	Yes

Total page faults = 10

Problem 5 – Optimal Page Replacement

Q: Same reference string, 3 frames, Optimal. Count page faults.

Ref	Optimal Evicts (farthest future use)	Frames	Fault?
1–3	–	[1,2,3]	Yes ×3
4	3 (next use: pos 10)	[1,2,4]	Yes

Ref	Optimal Evicts (farthest future use)	Frames	Fault?
1	–	[1,2,4]	No
2	–	[1,2,4]	No
5	4 (next use: pos 11)	[1,2,5]	Yes
1	–	[1,2,5]	No
2	–	[1,2,5]	No
3	5 (next use: pos 12)	[1,2,3]	Yes
4	3 (no more uses)	[1,2,4]	Yes
5	4 (no more uses)	[1,2,5]	Yes

Total page faults = 8 (minimum possible – the benchmark)

Problem 6 – Fixed Partitioning

Q: 100 MB RAM, OS=10 MB. Partitions: P1=20 MB, P2=30 MB, P3=40 MB. Processes: A=15 MB, B=25 MB, C=35 MB, D=18 MB. Which run? Internal fragmentation?

- A (15 MB) → P1 (20 MB): internal frag = 5 MB
- B (25 MB) → P2 (30 MB): internal frag = 5 MB
- C (35 MB) → P3 (40 MB): internal frag = 5 MB
- D (18 MB) → No free partition! (only 3 partitions)

Maximum 3 processes run simultaneously. Total internal fragmentation = 15 MB.

Problem 7 – Swapping Time

Q: Disk seek=8 ms, rotational latency=4 ms, transfer=40 MB/s, process=20 MB. Find swap time.

```
Transfer time = 20 MB / 40 MB/s = 0.5 s = 500 ms
Swap out = 8 + 4 + 500 = 512 ms
Swap in  = 512 ms
Full swap (out + in) = 1024 ms ≈ 1.024 seconds
```

Problem 8 – Working Set

Q: Reference string over 10 refs: 1,2,3,1,2,4,1,2,3,4. Window Δ=4. Find working set at end.

```
Last 4 references (positions 7–10): 1, 2, 3, 4
Working Set = {1, 2, 3, 4}   (size = 4 pages)
```

4.11 Quick Revision Notes

Key Formulas

```
Physical Address    = Base + Logical Address          (dynamic relocation)
Physical Address    = Frame × Frame Size + Offset     (paging)
EAT (demand paging) = (1 - p) × MA + p × PFT
EAT (TLB)           = HR × (TLB + MA) + (1-HR) × (TLB + 2×MA)
Page Number         = Logical Address / Page Size
Offset              = Logical Address % Page Size
Working Set W(t,Δ) = pages referenced in last Δ time units
```

Page Replacement – Quick Reference

Algorithm	Rule	Implementable?	Belady's Anomaly?
FIFO	Replace oldest page	Yes	Yes
Optimal	Replace page used farthest in future	No (benchmark)	No
LRU	Replace least recently used page	With hardware	No
Second-Chance	Circular FIFO with reference bit	Yes	No

Page Table Sizes

Address Space	Page Size	# Pages	Page Table Size (4 B/entry)
32-bit	4 KB	2^20 = 1,048,576	4 MB → needs multi-level PT
32-bit	4 MB	2^10 = 1,024	4 KB
64-bit	4 KB	2^52 (huge)	4 PB → must use multi-level PT

10 Key Points to Remember

11. Logical address is generated by the CPU; physical address is the actual RAM location.
12. The MMU performs address translation (logical → physical).
13. TLB caches page table entries for fast translation.
14. Internal fragmentation = waste inside an allocated block.
15. External fragmentation = waste between allocated blocks.
16. Paging: no external fragmentation, but has internal fragmentation (last page).
17. Segmentation: can have external fragmentation, but no internal fragmentation.
18. Virtual memory allows processes larger than physical RAM.

19. Demand paging loads pages only when accessed (on page fault).
20. Thrashing: page-fault rate is so high that CPU does almost no useful work.

4.12 Practice Questions

Multiple Choice Answers

Q	Topic	Answer
1	Address generated by CPU	b) Logical address
2	Algorithm causing many small holes	b) Best-Fit
3	Belady's anomaly	c) FIFO
4	Division of physical memory in paging	c) Frames
5	Thrashing occurs when	b) Page fault rate is very high
6	Optimal algorithm	b) Used as a benchmark
7	TLB hit means	b) Translation found in cache
8	No external fragmentation	d) Paging

Short Answer Topics (2–3 marks each)

21. Differentiate between logical address and physical address.
22. Explain the purpose of the MMU.
23. Explain internal and external fragmentation with examples.
24. What is a TLB and why is it used?
25. Compare paging and segmentation.
26. What is demand paging? State its advantages.
27. Define thrashing and state its causes.
28. What is Belady's anomaly? Which algorithm suffers from it?
29. Explain the working set model.
30. Differentiate between global and local page replacement.

Long Answer Topics (10–15 marks each)

31. For reference string 7,0,1,2,0,3,0,4,2,3,0,3,2 (3 frames), calculate page faults for FIFO, LRU, and Optimal.
32. Explain demand paging: page-fault handling steps and EAT calculation with examples.
33. Compare fixed partitioning, dynamic partitioning, paging, and segmentation.
34. Describe thrashing: causes, detection, and solutions. Include the working set model.
35. A system has TLB=1 ns, Memory=100 ns. At 98% hit ratio, find EAT. Repeat for 90%.

End of Module 4 Study Material

Reference: Silberschatz, Galvin & Gagne – Operating System Concepts (Ch. 8–10)