

Assignment - II

## 1. Address translation with multiple processes.

In modern systems with multiprogramming, multiple processes coexist in memory. Each process is given a logical (virtual) address space, while the actual data resides in physical memory (RAM). The translation is managed by memory management unit (MMU) with help of page tables.

Steps:

- (i) Logical Address Generation
- (ii) MMU translation via page table
- (iii) Physical Address formation
- (iv) Access to Physical Memory

## 2. Memory layout with Fragmentation

- Internal Fragmentation: Fixed partitions waste space inside blocks.

eg. Block = 8 KB, Process needs 6 KB  $\rightarrow$  2 KB wasted

- External Fragmentation: Free memory is scattered.

eg. layout:

[ 10 MB | Free 5 MB | 20 MB | Free 8 MB | 15 MB | Free 6 MB ]

Modern OS solutions (beyond compaction)

- Paging: Removes external fragmentation
- Segmentation + Paging: Reduces both types
- Buddy system: Split/merges memory in powers of two
- Sbb allocation: Efficient kernel memory use
- Virtual memory: Non-contiguous allocation avoids external fragmentation



### 3. Paging-based memory Allocation model.

- memory is divided into fixed size pages.
- Each process has a page table mapping virtual pages  
→ physical frames.
- MMU + TLB to handle fast translation; on TLB miss,
- eliminates external fragmentation (any free frame can be used)

#### Trade-offs

- memory overhead: large page tables per process;  
mitigates with multi-level tables.
- Speed: TLB hits are fast; misses / page fault slow down execution.
- no external fragmentation; but small internal frag.
- Huge pages: Reduce TLB pressure (faster) but increase internal waste.

4. When an OS manages a virtual memory, it works closely with the hardware to make sure processes run in isolated, protected, and efficient environments.

#### Interaction b/w Hardware and OS in virtual memory

- virtual to physical Address translation.
- Page tables
- Translation Lookaside Buffer (TLB)
- Memory Protection
- Page Faults and Demand paging
- Hardware support for isolation



## Key Hardware Structures

- MMU : handles address translation
- Page tables : OS managed structures mapping virtual pages to physical frames
- TLB : hardware cache for fast translations.

5. virtual address size = 16 bits  
 page size = 1 KB = 1024 bytes =  $2^{10}$  bytes (machine)  
 Page table entry (PTE) size = 2 bytes

(a) virtual add = 16 bits total

Page size =  $2^{10}$  → 10 bits used for offset within a page

Remaining bits for page no =  $16 - 10$   
 = 6 bits

So, page no. field = 6 bits

page offset field = 10 bits

no. of virtual pages =  $2^6 = 64$  virtual pages

(b) Each virtual page needs one entry in page table

no. of entries = 64

Each entry = 2 bytes

Page total size =  $64 \times 2 = 128$  bytes

## PART - B

### 6. Memory Allocation Simulation

A system has 1000 KB of free memory.

Process	Size (KB)
P <sub>1</sub>	212
P <sub>2</sub>	417
P <sub>3</sub>	112
P <sub>4</sub>	426



Using first-fit, best-fit and worst-fit

First-fit

Start = 1000 KB

Place  $P_1$  (212 KB)  $\rightarrow$  remaining =  $1000 - 212 = 788$  KB

Place  $P_2$  (417 KB)  $\rightarrow$  remaining =  $788 - 417 = 371$  KB

Place  $P_3$  (112 KB)  $\rightarrow$  remaining =  $371 - 112 = 259$  KB

Try  $P_4$  (426 KB)  $\rightarrow$   $259 \text{ KB} < 426 \text{ KB} \Rightarrow$  can't place

Result : Allocated  $P_1, P_2, P_3$  ; free = 259 KB

Best-fit : Same sequence here (only one at each step)  
 $\rightarrow$  259 KB leftover ( $P_4$  unallocated)

Worst-fit : same result  $\rightarrow$  259 KB leftover

Conclusion : All three produces identical results for this allocation . 259 KB unused  
•  $P_4$  not placed.

7. Page reference string :

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 and 3 frames

FIFO

frames	Ref	7	0	1	2	0	3	0	4	2	3	0	3	2
	$F_1$	7	7	7	2	2	2	2	4	4	4	0	0	0
	$F_2$	-	0	0	0	0	3	3	3	2	2	2	2	2
	$F_3$	-	-	1	1	1	1	0	0	0	3	3	3	3
	Fault	F	F	F	F	H	F	F	F	F	F	F	H	H

Total FIFO page faults = 10



## Optimal (Belady's algo)

Rref	F <sub>1</sub>	F <sub>2</sub>	F <sub>3</sub>	Fault
7	7	—	—	F
0	7	0	—	F
1	7	0	1	F
2	2	0	1	F
0	2	0	1	H
3	2	0	3	F
0	2	0	3	H
4	4	2	0	F
2	4	2	0	H
3	4	2	3	F
0	0	2	3	F
3	0	2	3	H
2	0	2	3	H

Total Optimal page faults = 7 (min. possible)

## LRU (Least Recently Used)

Rref	7	0	1	2	0	3	0	4	2	3	0	3	2
F <sub>1</sub>	7	7	7	2	2	2	2	4	4	4	0	0	0
F <sub>2</sub>	—	0	0	0	0	0	0	0	0	3	3	3	3
F <sub>3</sub>	—	—	1	1	1	3	3	3	2	2	2	2	2
Fault	F	F	F	F	H	F	H	F	F	F	F	H	H

Total LRU page faults = 9

Ordering (Best → worst) is :

Optimal (7) < LRU (9) < FIFO (10)



8.

Disk write time = 10 ms per page

memory write time = 100 ns per page

replaced pages that are dirty = 30%.

no. of replaced pages = 1000

(a) Additional time overhead due to dirty pages when replacing 1000 pages

dirty pg.s =  $0.30 \times 1000 = 300$

disk time =  $300 \times 10 \text{ ms} = 3000 \text{ ms} = 3 \text{ seconds}$

memory time =  $1000 \times 100 \text{ ns} = 100,000 \text{ ns} = 0.1 \text{ ms}$

Total time spent = disk time + mem

So, additional overhead = 3 sec.

total (disk + mem) =  $3000 \text{ ms} + 0.1$

=  $3000.1 \text{ ms}$

$\approx 3 \text{ sec.}$

(b) Proposed optimization to reduce this overhead.

Best single practical optimization:

Background pre-cleaning + prefer clean victims.

Two linked ideas that are commonly used together:

- page-cleaner daemon (background write-back)
- Clean first victim selection (enhanced clock / "modified list")



## 9. Autonomous Vehicle Case Study:

### (a) Working set model + replacement policy

- OS tracks recent active pages per task.
- For object detection: Allocate stable working set
- For inference: Allows flexible replacement as it adapts to available memory.

### (b) Memory Allocation Strategy

- User priority-based dynamic allocation
- Real-time responsiveness ensured by working set + real time schedule.