MOD-4 OS

(1)

* Memory management: track of status of each memory location, whether allocated or free. & allocate memory dynamically.

* Requirements in memory management → Reallocation, ✓
Protection, ✓
sharing, ✓
logical organisation,
physical organisation.

1) Reallocation → not possible in advn. to tell which program will reside where,
& not possible that when swapped back in, it resides at the same locatn.
→ After loading program into main memory, OS must be able to read logical & physical address.

2) Protection → There are multiple processes running, thus one program may start writing on other programs address space. Each process should be protected from such interference
→ as knowing the location of program while execution is impossible, thus protection should be satisfied by processor.

* More Relocation, more harder protection.

3) Sharing → several process needs same part of memory, thus each process has the same copy of required memory / program, rather than having the orginal. That file is loaded to main memory & copies are shared with all.

4) logical organisation → (modules) → Most of the programs are organised into modules, some are unmodifiable & some are modifiable & contain data. OS has some basic modules.
- Modules are written & ~~combined~~ compiled independently & references from one module to another are resolved by system at runtime.
- Each module has its own degree of protection.

5) physical organisation → main memory & secondary memory.
→ main issue is flow of data from ~~main mem~~ secondary memory to main memory & is impractical to understand this.
→ when memory is available but data is insufficient, modules are assigned to memory.
→ In multiprog., programmer does not know how much space will be available,

* Memory partition

1) Fixed partition → main memory is divided into fixed parts. (static)
& size of parts may vary & each parts holds a single process.
→ degree of multiprogramming = no. of partitions. = max no. of processes.
adv → easy to implement & manage
disadv → suffers internal fragmentation, (each process is contagious on one memory loc)
less multiprogramming..

2) Dynamic partition → each process acquires the space as much as it requires.
loc have variable size partitions & are not made at system generation.
adv → no internal fragmentation, simple & easy to manage, high multiprogramming.
disadv → External fragmentation eg → 2 spaces of 1mb & 2mb, & a process of 3mb, yet we cannot accomodate it.

3) Buddy system → (power of 2), (buddy block → dividing a block into 2)
→ memory is divided into fixed size of blocks, size is decided as 2^n
→ system finds smallest available block that can handle memory request
* these blocks of memory are interconnected using binary tree (smallest = leaves)
a smallest size block is found & if block is bigger, it is divided into 2 blocks (buddy block) & on work done, they are again rejoined.
adv → reduces fragmentation, fast allocation & deallocation
disadv → Internal fragmentation, 2^n size blocks, overhead.

4) Fragmentation → unwanted problem in OS.
→ processes are loaded & deleted from the memory, this generates free space in memory.
These small spaces cannot be allocated, thus memory is inefficiently used.
= fragmentation.

- Internal fragmentation → when process is allocated to a memory block, but size of process is smaller than memory, remaining space = waste
= this is internal frag.
- External fragmentation → there is enough memory space available for a request by a process but the memory available is not contagious.
↳ this is external frag.

adv → optimized storeage & less failure
disadv → regular defragmentation, slows the storeage.

5) Paging → eliminates need of contagious allocation of memory

→ process of retriving processes in the form of pages from secondary memory to main memory = paging. (Page = same size)

* seperate each process into pages, (processes are stored in pages)

→ physical memory is divided into fixed size blocks = pages, of same size as memory is requested, as allocated pages. page table is used to maintain this

→ paging reduces internal fragmentation (as pages are of very small size)

→ External fragmentation is there as memory is fragmented into many parts.

6) segmentation :→ A process is divided into segments of different size

Types → a) virtual memory segmentation → each process is segmented into no. of segments, but segmentation is not done and may or may not take place during run time,

b) simple segmentation → each process is divided into segments & all are loaded into memory at run time.

* segment table (Base address) (segment limits)

• maps 2D logical address to 1D physical address.

• Base address → it contains starting physical address where segment resides in mem

• segment limits → also known as segment offsets, specifying length of segments.

• segment no. → no. of bits required to represent segments.

• segment offsets → no. of bits req. to rep. size of segments

adv → no internal fragmentation, improves cpu utilization.

→ external fragmentation

q.12 # First fit → search through list of available memory block, (check for 1st box that can accomodate the requested memory)

→ once block is found, the block is split into 2, the used part & the free part.

* Best fit → search through list of available blocks, (find block that is closest to size that is required)

→ divided into 2 parts → used & unused. | Reduces fragmentation.

dis → overhead increased & high internal fragmentation

adv → less fragmentation, less external frag.

* Segment table → maps 2D logical [illegible] controller matches both address ③

* Worst Fit → travers through available blocks & find max size block avail ④ & place process in it.

→ this is a slow process, as to traverse everytime.

→ high internal fragmentation.

→ slow process, we need to traverse everytime.

* Next Fit allocation → similar to 1st fit, but (scan ahead from previously allocated space),

→ less overhead from 1st fit, as we do not need to traverse entire memory again.

→ reduced memory fragmentation & small gaps.

4.3

* Cache Memory organisation ⇒ mapping data in memory to location in cache.

→ cache memory is highspeed, small quantity of memory, used to copy & store data from main memory.

→ Act as a buffer b/w RAM & CPU, hold the requested data instruction, thus directly available on CPU.

CACHE Architecture (L1, L2, L3) ⇒

L1 → smallest & fastest (64 kb),

Each core in processor has its own L1

↳ L1 has 2 parts L1 DATA & L1 Instruction

↓ ↳ handles info to be fed to processor.

hold info to be written in to main memory

L2 ⇒ L2 is also embedded to each core of processor, & has more storeage than L1 cache but is slower than L1.

L3 ⇒ Not embedded to cores in CPU, (L3 acts as shared storeage pool that entire CPU can access.

⇒ only 2 times faster than RAM.

⇒ L3 reduced the chance of cache data miss.

4.9

Cache Memory organisation

- Cache memory is a high speed, small quantity of memory that stores copies of data from frequently used main memory locations.
- Cache memory acts as buffer bet^n RAM & CPU. and hold the requested data and instructions, so that they are immediately available on CPU.
- The cache organisation is about, mapping data in memory to a location in cache.

Cache Architecture:

- We have (L1, L2 & L3) architecture system.
- L1 cache → also known as primary cache, is the smallest and the fastest memory level (of size 64 KB)
  - Each core in processor, has its own L1 cache Built in.
  - L1 has further 2 levels L1- instruction & L1- Data.
  - L1 - instruction handles info. fed from processor.
    L2 - Data hold info to be written to main memory.
  - L1 - cache can transfer data at max speed, making CPU efficient.

* L2 cache → secondary memory cache, it is also embedded with each core of processor.
- It has more storeage than L1 cache, but has slow speed than L1, yet faster than RAM.

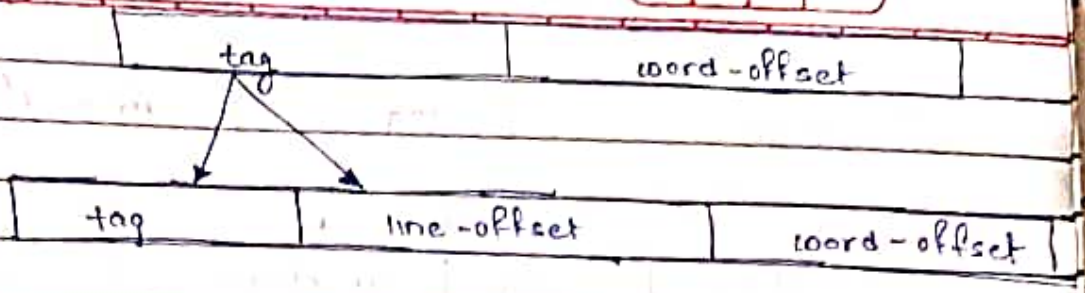
* L3 cache →
- This is not embedded to each core of CPU, L3 acts as a shared storeage pool that the entire processor can access.
- This is only twice as fast as RAM,.
- If CPU fails to find memory, it needs to find that in slower memory systems, thus is called cache miss.
- The introduction of L3 cache reduced the chance of a miss and helped improve performance.

* Address mapping

* There are 3 different types of cache mapping:

1) Direct mapping → Simplest method, it maps each block of main memory to only one possible cache line. (each memory block to specific line in cache)
- If the line is previously taken up by memory block, when a new block needs to be loaded, the old block is trashed.

- Address space is split in two parts, index field & tag field. The cache is used to store tag field. & rest is stored in main memory.



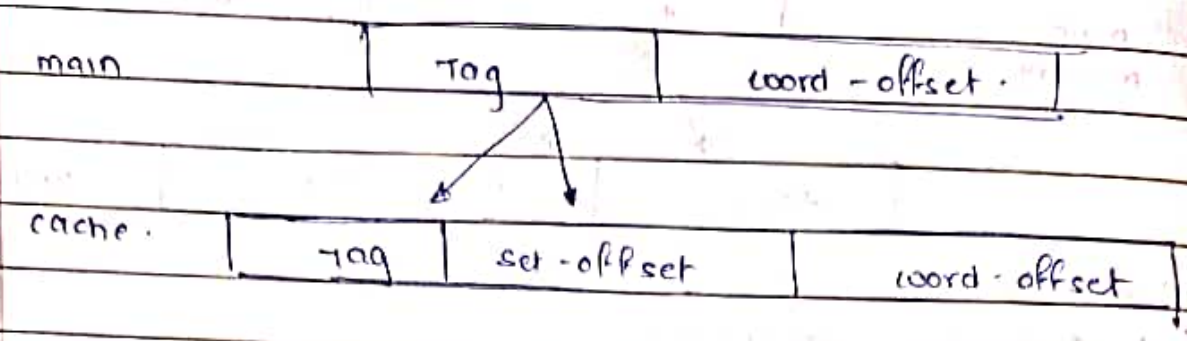
2) Associative mapping:

- Here associative memory is used to store the content & address of the memory word.
- Any block can go into any line of cache
- This enables the placement of any word at any place in the cache memory.
- It is considered to be the fastest & flexible form.
- Here, indexed bits are zero.

3) Set - Associative mapping:

- This is the enhanced form of direct mapping, where drawbacks from direct mapping are removed (possibility of thrashing is removed)
- This is done by, having exactly one line that a block can map to in the cache, we will group a few lines together creating a set.
- Then a block in memory can map to any of line of the specific set.
- Set - A allows, each word in cache to have one or more words in main memory with same index value.

- set A. cache mapping combines best of direct & associative,
→ mapping the index bits are given by set - offset bits.



Cache coherence:

- This issue occure from concurrent operations from several processors & various cache may hold identical memory blocks.

- Cache coherence has three levels:
  - Each writing operation seems to happen instantly.
  - Each operands value & changes is seen in every processor in precisely the same order.
  - Non-coherent behaviour results from many processors interpreting same actions in various ways.

Methods to resolve cache coherence:

Two methods.

a) write through → every memory operation updated the main memory. If word is present in the cache memory at the requested address, cache memory is also updated simultaneously with main memory.
- RAM & cache always hold the same information, systems with direct memory access transfer, the quality is crucial. To make sure information in main-memory is upto date all the time.

adv
- It provides highest level of consistency.
- It requires greater no. of memory access.

(b) Write back :

- only the cache location is changed during the write operation.
- When the coord is withdrawn from the cache, the location is flagged, so it is replicated in memory (main).
- This approach was developed beoz, words may be updated numerous times while they are in cache. But, as long as they are in there it does not matter, whether copy stored in main memory is outdated as request is fulfilled.

adv • a small no. of memory operations accessed & write operation.

disadv • Inconsistency.

* Cache Coherence Protocols :

(a) MSI → a fundamental protocol used in multiprocessor system

M → Modified ⇒ data in cache is incompatible with main memory, thus the block is been updated in cache. Thus, when data removed from cache, the cache is responsible for writing data changes to main memory.

S → Shared → Atleast one cache has atleast one copy of block, that is not been update

I → Invalid → If the block is going to be stored in this cache, it must be obtained from RAM.

(b) MOSI ⇒

O → Owned → It is used to signify the ownership of current processor to this block & will respond to inquires if another process wants this block.

(c) MESI

E → Exclusive ⇒ exclusive signifies that the data is clean & the cache & main memory have same data.

S → Shared ⇒ other caches on computer may also hold same cache line.

(d) MOESI - (Javatpoint)

Types of coherence

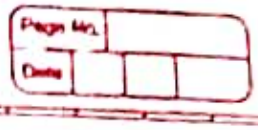
(a) Directory based ⇒

- A directory based system keeps the coherence amongst the caches by storing shared data in single directory.
- In order to load data from primary memory into cache, processor must request permission through directory.
- Directory, either upgrades or devalues other caches that contain the record when modified.

(b) Snooping ⇒

- Individual cached watch address line during snooping to look for access of memory locations that they have cached. A write invalidate protocol is what it is known as.
- When a write activity is seen to a memory address, for which cache maintains a copy, the cache controller invalidates its own copy of snooped memory location.

Software program that acts as intermediary betn user & providing an user friendly env. to interact ....



(c) Snarfing →

- A cache controller, uses this method to try & update its own copy of memory location, when the second master alters the place in main memory by keeping an eye on both address and contents.
- The cache controller updated its own copy of underlying memory location with new data, when a corresponding is detected to a place of which cache holds copy.

* Swapping

- A memory management scheme, where any process can be temporarily swapped from main memory to secondary memory. So that main memory is available for processes. In secondary memory, place where swapped process is stored is swap space.
- The purpose is to access data present in hard disk & bring that to RAM, swapping is only done when data is not present in RAM.

* We get 2 more concepts.
  - swap-in → removing processes from RAM & adding them to hard disk.
  - swap-out → removing process/program from hard disk & put that back to RAM.

* Adv (swapping)
  - Helps CPU manage processes with single main memory.
  - helps creating & using virtual memory.
  - allows CPU to do multiple task simultaneously.

occurs when a program attempts to access the data or code in its address space but is not currently located in system RAM.

Page No.
Date

(*) DISADV (Swapping)

- If computer system looses power, user may loose all information related to program in case of substantial swapping.
- If the swapping algo. is not good, the composite method can increase no. of page faults & decrease overall processing performance.

Thrashing

- Thrash is poor performance of ~~main~~ virtual memory (paging) system. where some pages are being loaded repeatedly due to lack of main memory to keep them in memory.
- Thrashing occurs when computers, virtual memory resources are overused, leading to constant state of paging & page faults.
  leading to collapse of computer system.
- This may keep on going until user closes some running application or active processes free the ~~processor~~ virtual mem. resources.
- Thrashing is when page fault & swapping occurs very frequently at a very high rate, the OS has to spend more time swapping these pages. Thus CPU utilization is reduced.

* basic concept → if a process is allocated too few frames, then frequent page faults will be seen.

* Adv

- fast data writes → fast reorganizing data.
- Fewer failures → insufficient space may lead to failure.
- optimized storeage.

* disadv

- a need for regular defragmentation
- slowing read times, as the storeage is fragmented.

5) Paging :

- Paging eliminates need of contagious allocation of physical memory

(1) • Process of, retrieving processes in the form of pages from the secondary storage to main memory. is known as paging.
• Basic purpose of paging is to seperate each procedure into pages.
Frames will be used to split the main memory.

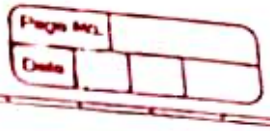
(2) • In paging, physical memory is divided into, fixed sized blocks called page frames, that are of same size as page used by process. When process requests memory, OS allocates one or more pages frames to process & maps processes logical pages to physical pages
• A page table is used to maintain mapping.

Page No.
Date

Size of Page Table :

- Part of process being executed by the CPU must be present in main memory during that time period.
- The page table must also be present in main memory all the time because it has ~~##~~ entry of all pages.
- Size of pg. table depends on no. of entries in table & bytes stored in each entry.
- Pg. table may also be considered as collection of frames ~~of~~ & stored in different frames OR in a single frame too.

providing an user friendly env. to interact ... intermediary both user & har



- To overcome this problem, high speed cache is set up for page table ent page table entries called as Translation lookaside buffer (TLB)

- TLB is a special cache, used to keep track of recently used transactions.
  It contains pg. table entries that have been recently used.

- Given a virtual address, processor examines the TLB, if page table entry is present, the frame no. is retrieved & real address is formed.
  If pg. table entry is not found in TLB, the pg. no. is used as index while processing the page table.

(TLB Hit)

(TLB miss)

- TLB 1st checks if page is already in main memory, if not a page fault is issued.

In TLB Hit

- CPU generates a virtual address
- Now, this is checked in TLB (present).
- corresponding frame is ~~received~~ retrieved, telling where main memory page lies.

In TLB Miss

- CPU generates a logical address.
- It is checked in TLB (not present)
- Now the pg. no. is matched to pg. table residing in main memory
- corresponding frame is retrieved, where main memory pg. lies.
- The TLB is updated with new PTE.

frame no. ⇒ address of main memory, where we want to refer.

Page No.
Date

Adv

* It helps in reducing memory space.
- It takes longer lookup time, as memory look up takes place concerning the virtual address.
- It makes page swapping easier.

Translation lookaside buffers

- Intro
  → In O.S. for each process we have page Tables, that consists of page table entries (PTE), PTE contains info like frame number, etc.
  PTE tells, where in main memory is actual page residing.

- The problem was to fast access data, & initially for that page tables were stored in registers.

- The idea used here is to place, page table entries in registers, for each request generated from CPU, it will be matched to appropriate pg. no. of page table, which will now tell where in main memory the corresponding page resides.

Everything is right, but issue is register is of small size & process may be big sized, thus not the practical approach.

The entries entire pg. table was kept in main memory to overcome, the size issue, but we require 2 references
(1) for find frame no.
(2) go to address specified by frame no.

acts as intermediary betn use
Providing an user friendly env. to interact ....

Page No.
Date

4) Hashed Pg. Table :

- Used to handle address spaces of >32 bits.
- In this, the virtual page, the no. is hashed to pg. table.
  → Pg. table mainly contains a chain of element, hashing to same elements.
- Each element has a;
  → virtual page no.
  → value of mapped pg. frame.
  → Pointer to next element.

5) Clustered Pg. Tables

- Similar to hash table, but each entry refers to several pages (i.e. 16) rather than 1.
- Mainly used for sparse address spaces where memory references are non-contagious & scattered.

Inverted Page table :

- Inverted page table is a global page table, maintained by OS for all processes.
- In Inverted PT. the no. of entries = no. of frames in main memory.
- There is always a space reserved for page, regardless of fact that it is present in main memory or not. leading to wastage of space.
- We can save the pages by inverting the pg. table, we can save details of pages only; present in memory.

Frame = index & info. saved inside block = process Id & pg. no.

* Techniques of structuring page table:

1) Hierarchical paged paging: (multilevel paging)

- There may be a case of page table too big to fit in contiguous space, thus we have a hierarchy.
- Here, logical address space is broken into many pg. tables.
- here, 2-level OR 3-level paging can be used.

2) Two-level pg-table

- consider a system of 32 bit page logical address space;

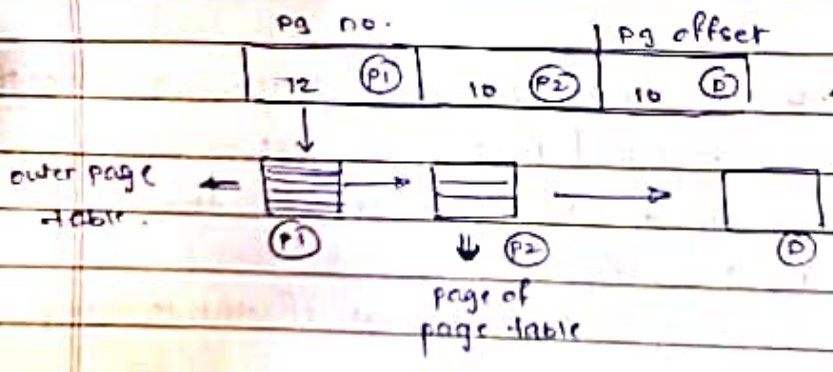
→ pg no. = 22 bits

pg offset = 10 bits.

and we have page table, that is further divided as;

→ pg. no. = 12 bits

pg offset = 10 bits.



3) Three level Pg. Table.

- for 64 bit logical address, we cannot have a large 2D page table, thus we use,

2nd outer PG	outer PG	inner PG	offset
P1	P2	P2	d
32	10	10	12