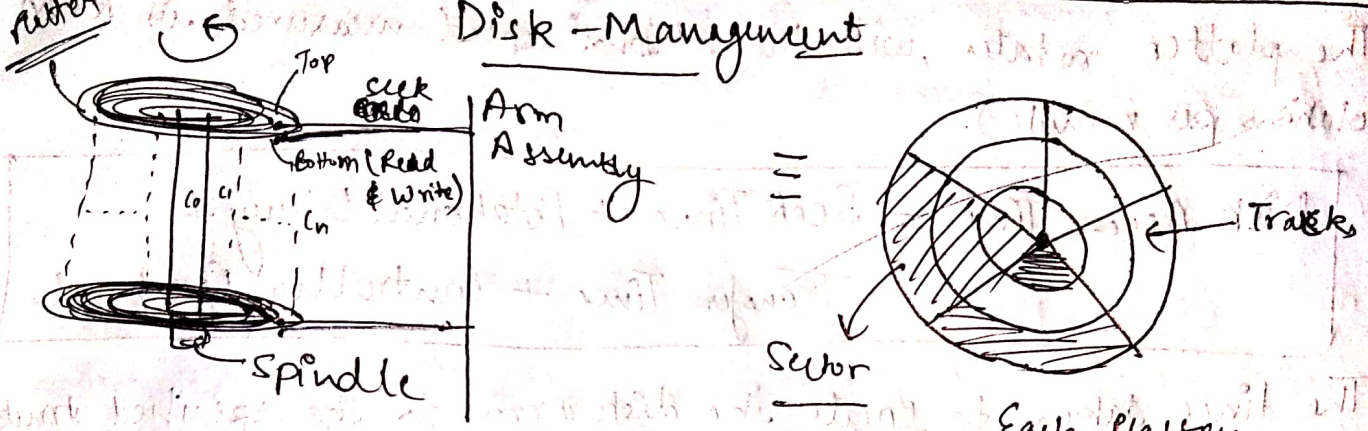
13/11/019

Module - III

- Memory management
- I/O "
- Disk "
- File System Management.

Disk - Management



Each Platter 2 surface

* Bit density (Data storage Density) — is high in Innermost track is low in outer most track.
* Each track sector stores same amount of data.
* Read Write Arm at both Top & Bottom surfaces separately at each Platter.
* Arm Assembly move only to & fro (It can read data of any track of any sector of all platter at once)

* Disk are tertiary / secondary storage system in memory hierarchy. It is an offline storage and data stored on disk are generally permanent.

* Disk Storage Mech.

- A disk contains multiple platter with 2 surface per platter. Each surface is divided into tracks and sectors and read-write head. R-W head has to-&-fro motion to read the data from desired track sector.
- The platter rotates, with the rotational speed measured in RPM (Rotations per minute).

> Disk Access Time = Seek Time + Rotational Latency + Transfer Time + Controller Overhead

- The time taken to locate the disk arm to specified track from where the data is to be read/write is Seek Time.
- ~~The time taken by~~ Seek Time depends upon - no. of tracks & speed of seek (arm)
- The time taken by desired sector of the disk to rotate into the position, so that it can access R-W head is Rotational Latency
- Rotational Latency/Time depends upon bandwidth (RPM) and distance from R-W Head of the sector to be take

14/11/019

- The time takes to transfer the requested data is known as Transfer Time (depends amount of data (block of data))

Note - The average rotational latency for a disk is 1/2 the amount of time it takes for the disk to make one revolution

Ex. 72,000 RPM, then 120 Rotations per second

i.e 1 Rota - $\frac{1}{120}$ second, ∴ Avg R. Latency = $\frac{1}{240}$ s

Q. Consider typical disk, that rotates at 15,000 RPM and has a transfer rate of 5×10^6 Bytes/second. If the avg. seek time of the disk is twice the Avg. R. Latency and controller's transfer time is 10 times the disk transfer time, the avg. time to read or write - 512 Bytes sector of the disk will be.

⇒ 15000 RPM ⇒ $\frac{15000}{60}$ = 250 per second.

i.e 1 Rotation = $\frac{1}{250}$ second.

Avg R. Latency = $\frac{1}{500}$ seconds = 2 milliseconds

Seek time = 2 × Avg R. Latency = $\frac{1}{250}$ seconds = 4 millisecond

Controller's transfer time = 10 × Disk transfer time

= $10 \times \frac{512}{5 \times 10^6}$ seconds

Transfer time = $\frac{512}{5 \times 10^6}$ second

= 102.4 μseconds

Disk Access Time = $\frac{1}{500} + \frac{1}{250}$

Controller Overhead = 10 × Disk transfer time

= 1024 μseconds. = 1 millisec.

Disk Access Time = 2 millisecond + 4 msec + 102.4 μsec + 1024

= 7.1024 milliseconds

• Disk Scheduling — FCFS C-SCAN
SSTF LOOK
SCAN C-LOOK

Why?

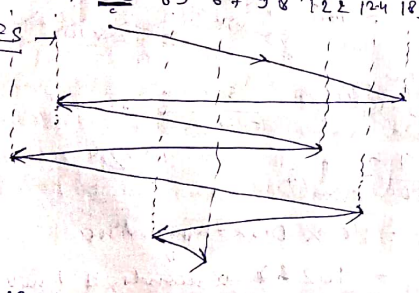
1. Multiple I/O request by different process, only one can be serv at a time, hence scheduling required.
2. Effective arm movement
3. HDD are slower devices hence need to be accessed in effective manner

Hard Disk Drive

* Consider a disk Q with the request for I/O to block on cylinders. R (98, 183, 37, 122, 14, 124, 65, 67). Currently the R-W head is at cylinder no. 53. Calculate the total arm movement under following — with 200 cylinders in total.

⇒ 0 14 37 53 65 67 98 122 124 183 199

(i) FCFS →



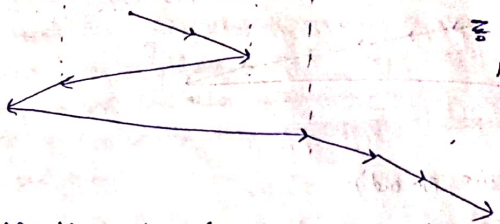
Total ARM Movement → (98-53) + (183-98) + (183-37) + (122-37) + (122-14) + (124-14) + (124-65) + (67-65)

⇒ 45 + 85 + 146 + 85 + 108 + 110 + 59 + 2

⇒ 640

(ii) SSTF (see the adjacent differ) (Shortest Time First)

0 14 37 53 65 67 98 122 124 183 199

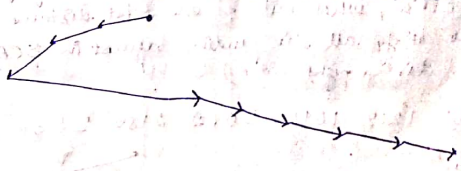


Total ARM Movement → (65-53) + (67-65) + (67-37) + (37-14) + (98-14) + (122-98) + (124-122) + (183-124)

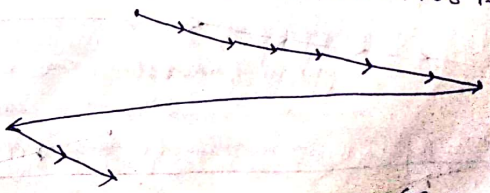
= 12 + 2 + 30 + 23 + 84 + 24 + 2 + 59 = 208

* SCAN → (Bi directional, 1 end) ≡ Elevator (default upward Algorith. if arm movement not given)
* C-SCAN → (Uni direct^n, Both end)
* LOOK → (Bidirectional, No end)
* C-Look → (Unidirectional, No-end)

3) SCAN 0 14 37 53 65 67 98 122 124 183 199



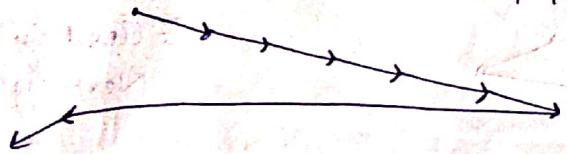
4) C-SCAN 0 14 37 53 65 67 98 122 124 183 199



TAM → (65-53) ... (199-183) + (199-0) + (37-0) + (37-14)

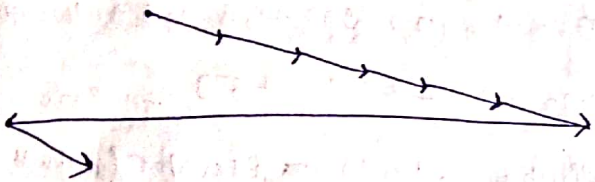
5) Look (Bidirn, No end)

0 14 37 53 65 67 98 122 124 183 199



6) C-Look (Unidirn, No end)

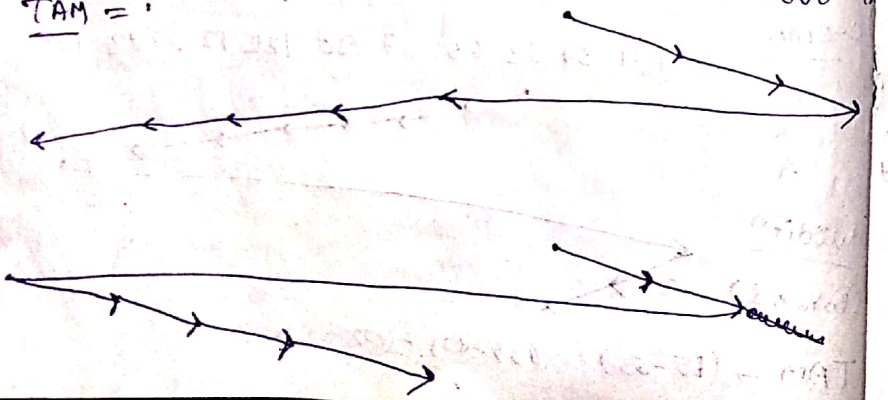
0 14 37 53 65 67 98 122 124 183 199



Q. Suppose, a disk drive has 5000 cylinders (0–4999). The drive is currently serving a request at 2150 & previous request was cylinder 1805. The queue of pending request in FIFO is:- 2069, 1212, 2296, 2800, 544, 1618, 356. Starting from the current head position, what will be the total distance the disk arm moves to satisfy all the pending request in SCAN & C-Look algorithm. (dirn → right side)

⇒ • 0 356 544 1212 1618 2069 2150 2296 2800 4999

SCAN TAM =

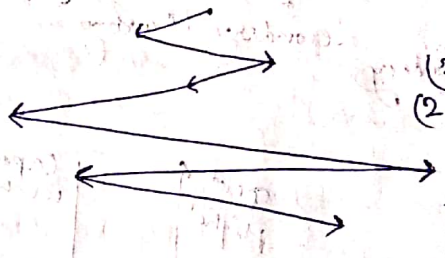


C-LOOK

TAM

Q. Consider, a disk, with the pending request queue → (10, 22, 20, 2, 40, 6, 38). Currently R-W Head is at cylinder '20'. Calculate the total seek time for the arm movement by applying FCFS scheduling and seek time per cylinder is 6 ms. Also, the disk rotates 20 revolutions per second, and has 100 word/sector. Each track of the disk has capacity of 400 words, then calculate the total time required to access one sector.

⇒ 20 2 6 10 20 22 38 40 (again in 20, Only in FCFS)



$(20-10)+(22-10)+(22-20)+(20-2)+(40-2)+(40-6)+(38-6)$

$= 10 + 18 + 2 + 18 + 38 + 34$

$= 100 + 12 + 34 = 146$

Seek time / cylinder = 6 ms

Total seek time = $146 \times 6 = 876$ ms

(Here controller overhead = 0)

* If 20 Rev/sec → 1 Rev = $\frac{1}{20}$ sec ⇒ Rotational latency = $\frac{1}{40}$ seconds = 25 msec.

Track capacity → 400 words & 100 words/sector

⇒ 1 Track contains 4 sectors

& 1 revolution = 1 Track complete = 400 words,

& 1 second = 20 rev = 8000 words/seconds ⇒ Transfer Rate

& If 8000 word → 1

(100 word → $\frac{1 \cdot 1000}{8000}$ = 12.5 m/sec. Ans

⇒ 1 sector (100 words) are transferred with 12.5 m/s. Ans

20/11/019

Memory Management

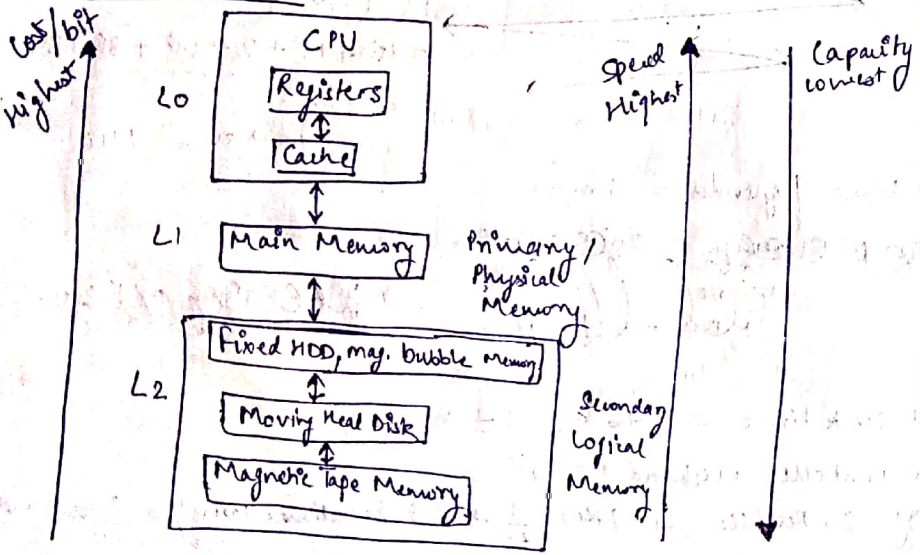
- Memory Hierarchy –

* For processing – Secondary to Main & CPU

* For Input/Output – To secondary (storage)

CPU
↕
Register
↕
Cache
↕
Main Memory
↕
Secondary Memory

Processing



- At any level of the memory, a memory can interact only with its immediate neighbour (i.e secondary can never interact with cache)

– Any system performance is calculated by miss ratio.

- Access time (t_i)
- Memory Size (S_i)
- Cost (C_i)
- Bandwidth (B_i) → Rate of transfer
- Unit of Transfer (X_i)
  → Block size transfer

⇒ All are for an i^{th} level of memory. (Parameters)

- $t_{i+1} > t_i > t_{i-1}$ (L1>L0)
- $S_i > S_{i-1}$
- $G_i < G_{i-1}$
- $B_i < B_{i-1}$
- $X_i > X_{i-1}$

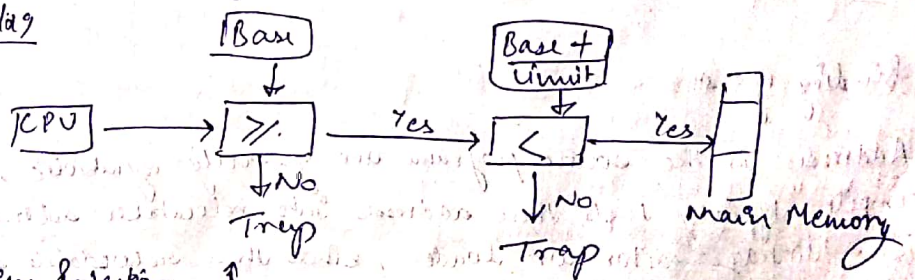
– A memory is a storage device, used for storing memory words (data & instructions), each on unique addresses (location).

– Memory consists of large array of word or byte, each with its own address and CPU fetches instruction from memory according to the value of program counter.

– After the instruction has been executed, on the operands + results are stored back into the memory.

– For successful execution of process, the OS needs to ensure that each process has separate memory space. To separate memory spaces, the range of legal addresses that a process may access must be determined and must be ensured that the process can access only these legal addresses.

23/11/19

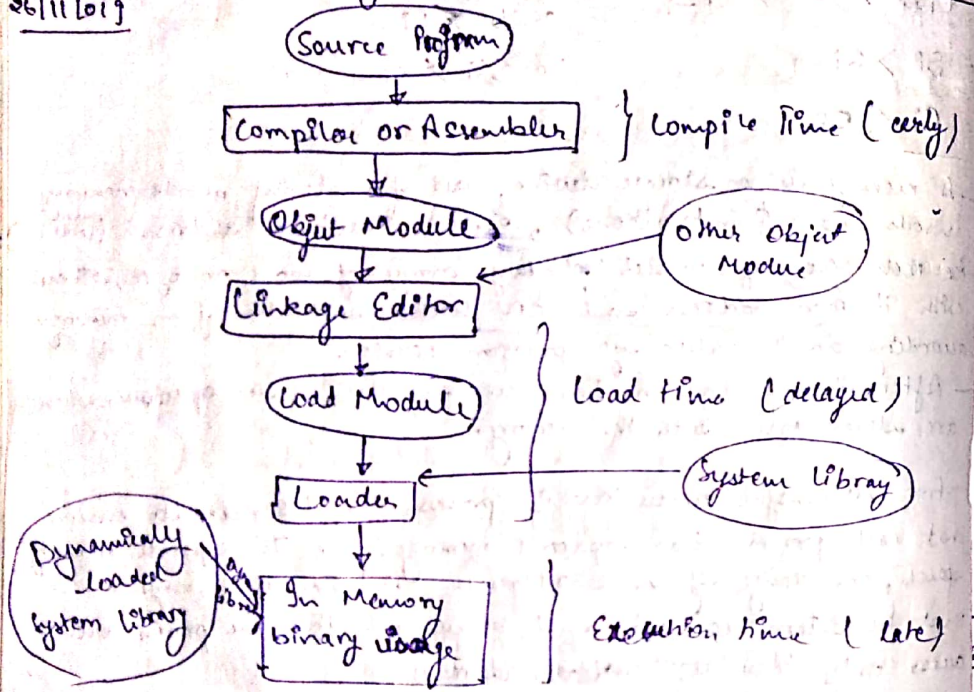


System Protection → Base & Limit

Address Binding: (Binding of Address to the data or Instruction)

• Run karne se pehle pata hai ki program kahan jayega →
  - compile time binding
  - load time binding
  - run time binding

26/11/019



• Binding of :

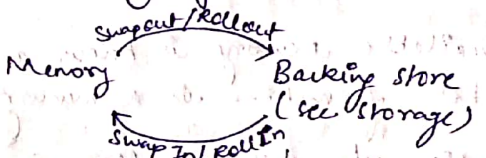
• Addresses in the source program are generally symbolic. A compiler typically binds these addresses into relocatable addresses. The linkage editor or loader, binds these relocatable addresses into absolute addresses. Each binding is a mapping from one address space to other. (early binding, delayed binding, late binding)

• The address generated by CPU is known as logical address whereas the corresponding address seen by the memory unit is physical address.

— The set of all logical addresses of a program is known as logical address space. All the corresponding physical address of that program, is known as physical address space.

Swapping

— During execution, the process must be brought into memory for execution. A process however can be swapped out temporarily to a backing store (secondary storage) & can be brought again to the memory for continued execution.



→ Swapping increases degree of multiprogramming.

27/11/019

Memory Allocation

(I) Contiguous M.A (consecutive)

(II) Non-Contiguous M.A (non-consecutive / seperate)

(i) Assign the consecutive block of memory for process requesting for memory

(ii) Assigns the seperate memory block at different location in memory space in a non-consecutive manner to a process requesting for memory.

Contigous

- Consecutive blocks of memory are allocated
- Donot have the overhead of address translation while execution
- Process executes faster.
- The memory space must be divided into fixed sized partition (equal / variable), and each partition is allocated to a single process only.

Non-Contigous

- Separate blocks of memory are allocated
- Have overhead of address translation while execution
- Process doesn't executes faster.
- Divide the process into several blocks and place them in different parts of the memory according to the availability of memory space.

* Hole -

- The strip or chunk of available memory is known as hole.
- There may be multiple hole formation in a memory.

⇒ Fixed Size Partition Allocation v/s Dynamic Memory Allocation

FSPA

1) Memory is divided into fixed partition of equal or varying size
2) Each partition contains exactly one process. The partitions cannot be easily expanded or shrunk.

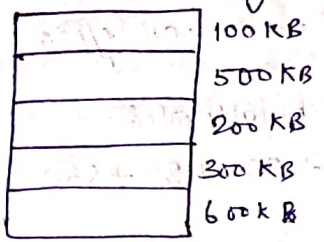
DMP

1) Initially, all memory is available & is considered as one large block of memory or hole.
2) All memory is available for user process.
3) User-process has to return the allocated memory once, it finishes its execution.
4) Memory can be returned in any order without any relation to the order in which it was allocated, hence multiple holes will be formed.

Memory Allocation Strategies

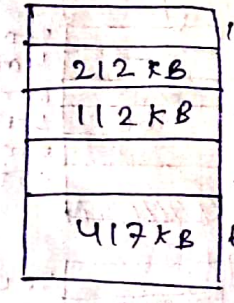
(i) First fit
(ii) Best fit
(iii) Worst fit
(iv) Next fit

Q) Given 5 memory partitions of 100, 500, 200, 300 & 600 KB in order. How would the (i) First (ii) Best (iii) Worst (iv) Next fit algorithm places the process of size 212, 417, 112, 426 KB in order. Which algorithm makes the most efficient use of memory?



Process: 212 KB, 417 KB, 112 KB, 426 KB

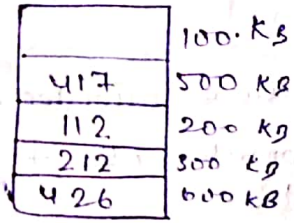
(i) First fit



426 KB cannot be allocated.

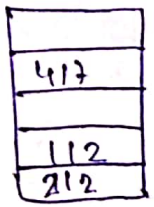
⇒ Hole (Total memory wastage)
= (100 + 288 + 88 + 300 + 183) KB = 959 KB

(ii) Best fit - Where memory wastage is least



Memory wastage = 100 + 83 + 88 + 88 + 174
= 533 KB

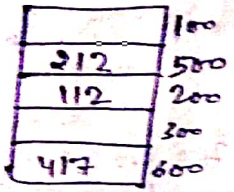
(iii) Worst fit - Memory wastage is maximized.



= 100 + 83 + 200 + 188 + 388
= 959 KB

426 X

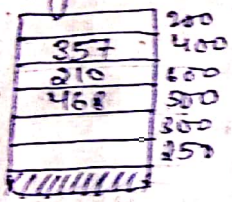
(iv) Next fit → In a cycle



wastage = 100 + 288 + 88 + 300 + 183

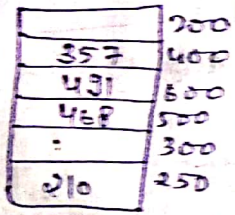
Q. Given six memory partitions: (200, 400, 600, 500, 300, 250) KB
Process: 357 KB, 210 KB, 468 KB, 491 KB in order.
• Apply all strategies & determine which algo make most efficient use of memory

(i) First



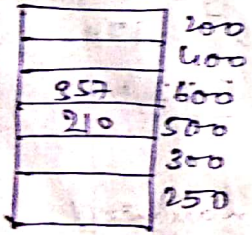
491 KB cannot be allocated.
= 200 + 43 + 390 + 32 + 550
= 1215 KB

(ii) Best



wastage = 200 + 43 + 109 + 32 + 300 + 40
= 724 KB

(iii) Worst



468 & 491 can be allocated.
wastage = 200 + 400 + 550 + 243
= 1683 KB

(iv)

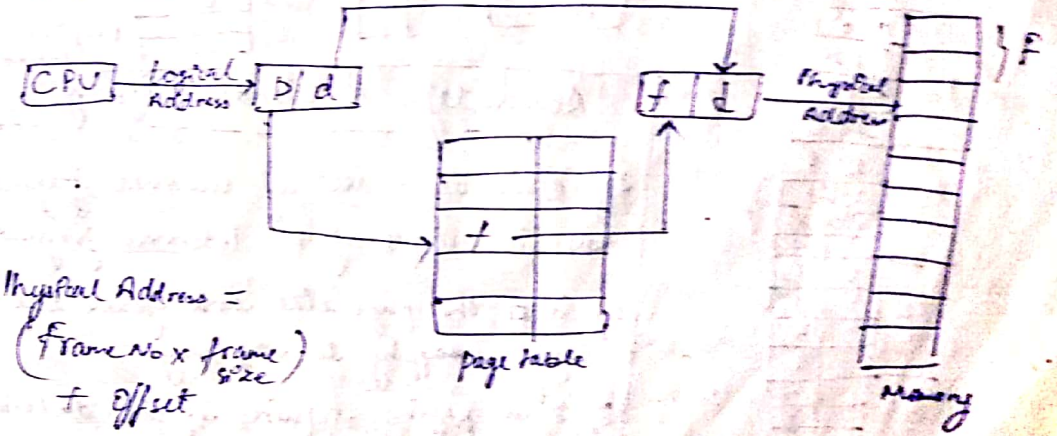
30/11/05 **Internal Fragmentation v/s External Fragmentation**

• When a process is allocated, more memory than required, & space is left unused & it occurs, when a memory is divided into fixed size partitions.
– It can be used by allocating memory dynamically or having partitions of different sizes (unlike system)

External
→ Many small, non-contiguous blocks of unused space are found, which can serve a new request, if all of them are joined together, but as they are not adjacent to each other, a new request cannot be served.
→ It can be used by → compaction, paging & segmentation.

Compaction
– External frag. can be reduced by compaction or shuffling the memory content & to place all free memory (holes) together in one large block.
– It require large amt of relocation & hence, it is not feasible.

Paging



Physical Address =
(Frame No x frame size) + Offset

p = page No.
f = frame No.
d = Offset / displacement

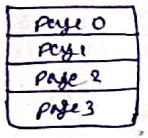
• Paging is a memory management technique, which divides main memory into fixed size partitions called frames & divide logical memory into same size partitions called Page

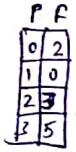
- The size of partitions is usually in power of 2. (Read Galvin)

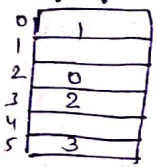
- Page table is a data structure maintained by OS to represent page frame mapping and resides in main memory.

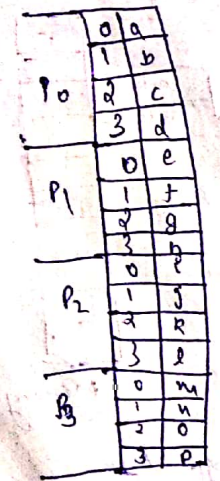
• The size of frame is kept equal to the size of page to have optimum utilisation of the main memory and to avoid external fragmentation.

• Page address is called logical address and is represented by page no. 'p' and page offset 'd'. Similarly, frame address is called physical address and is represented by frame no. 'f' and frame offset 'd'.









Advantages & Disadvantages of Paging

1) Paging is a sol^n of external fragmentation but it suffers from internal fragmentation.

2) Since the page size and frame size is equal swapping is very easy.

3) Since paging requires special data structure called pagetable which resides in main memory hence paging is not suitable for small memory sizes.

Page Table

⇒ No. of page entries = no. of pages in program / logical memory

Each process have its individual pagetable.

Each pagetable resides in main-memory (additional overhead of memory)

• Each pagetable has PTBR (Page Table base register) value & PTLR (Page Table length register) value. (store in PCB).

• Page Table is a metadata.

• To access any instruction two memory cycles are required. One for PageTable and other for actual content hence access time is doubled and speed is half.

* Memory is byte addressable : 1 Byte store (default)

* Memory is 2 byte addressable : 2 Byte store.

Note - Consider physical & logical memory which is byte addressable

Q. If logical address = 31 Bit.

⇒ Logic Address Space ⇒ $2^{31} \times \underset{(1\ Byte)}{1} = 2$ GB (1 Giga Byte = 2^{30} Byte)

Q. logical address = 128 Mega Byte. of space. Find, how many bits are required for logical address.

= $128 \times MB = 2^7 \times 2^{20}$ Bit. $\Rightarrow 2^{27}$ Bit $\Rightarrow$ 27 Bit.

Q. Physical Address = 24 Bit $\Rightarrow$ P.A Space = $2^{20} \cdot 2^4 = 16$ MB

Q. 14 Bit address (1 Byte) = $2^{14} \times 1 = 2^{14}$ byte = 16 KB (Total size of Memory)

22 Bit address (2 Byte) = $2^{22} \times 2 = 8$ MB

34 Bit address (4 Byte) = $2^{32} \times 2^2 = 64$ MB.

How many bits required.

Q. 64 KB memory size (1 Byte) = 64 KB / 1 Byte = $64 = 2^{16}$ ⇒ 16 bits
32 KB " " (1 Byte) = 32 KB / 1 Byte = 15 bits)
256 KB " " (2 Byte) = 256 KB / 2 Byte = 19 bits.
16 GB " " (4 Byte) = 16 / 4 ⇒ $4G = 2^{32}$ ⇒ 32 bits.

Q. Consider a memory scenario in which logical addresses bit are of 24 bits & physical address are of 16 bit and the page size is 1 KB. Calculate the bit required for. [p | d] [f | t]
also calculate total no. of pages and frames in memory.

Sol^n → Logical Address = 24 bits, page size = 1 KB = 10 bits.

Size of logical Memory = 2^{24} = 16 MB
Total no. of pages = $\frac{16\,MB}{1\,KB} = 2^{14}$ ∴ p = 14, d = 10

Size of Physical Memory = 2^{16} = 64 KB
Total no. frames = $\frac{64\,KB}{1\,KB} = 2^6$ = 6 bit ∴ f = 6, d = 10

Q. Logical Add = 33 bits, Physical Add = 24 bit.
Page size = 2 KB

Size of logical Add = 2^{33} = 8 GB
Total no. of pages = $\frac{8\,GB}{2\,KB} = 2^{22}$ ∴ p = 22 (pages), d = 11 (offset)

Size of physical Add = 2^{24} = 16 MB
Total no. of frames = $\frac{16\,MB}{2\,KB} = 2^{13}$ ∴ f = 13 (frames), d = 11 (offset)

04/12/2019

Q. Consider a logical memory of 256 pages and page size of 4 KB & physical memory of 64 frames. How many bits required for logical and physical address.
(one byte addres (default))

⇒ Total logical memory = $256 \times 4 \times 2^{10}$ Byte = $2^8 \times 2^2 \times 2^{10} = 2^{20}$ Bytes
= 1 MB = 20 bits. $\left(\frac{1\,MB}{1\,Byte}\right)$

⇒ Physical address = $\frac{1\,MB}{16}$ $64 \times 4\,KB = 2^8 \times 2^{10} = 2^{18}$
⇒ 18 bits.

Note - ① A special data structure called frame table is maintained by MMU to indicate the free & available frames in the main memory.

② To protect the memory from illegal physical address generation, one additional bit (valid / invalid) is attached to each entry of the page table. Before calculating the physical add. of corresponding logical address, the valid / invalid bit is checked. If the reference is set to valid, P.A will be generated else the reference will be trapped because of wrong address generation.

05/12/2019

Translation lookaside Buffer (TLB)

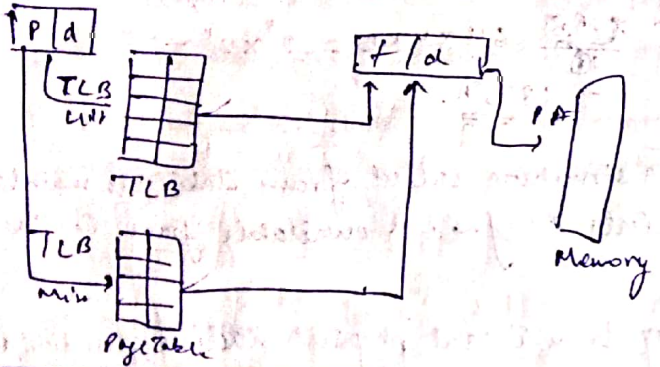
⇒ In paging, two memory cycle are required to access any instruction. With this approach, the access time of instruction increases drastically to those which causes major delay & this delay is intolerable under certain circumstances.

⇒ To reduce this access time, by reducing the effective memory cycles, a small path lookup hardware cache called TLB is used. The TLB is associative high speed mem. & each entry of TLB consist of two values - page no. & frame no.

- If a page is referenced by CPU, under TLB implemented paging scheme -

- If the page frame mapping is found in TLB
  - TLB Hit = No need to access page table. Only one memory cycle required.
  - TLB access Time + Memory Access Time

- If the page frame is not found in TLB
  - TLB Miss: Page table will be accessed to obtain page frame mapping. Two memory cycle required.
  - Time = TLB access Time + 2 × Mem. Acc. Time



Effective Memory Access Time

$EMAT$ = Hit of TLB × (TLB access Time + Memory Access) + Miss of TLB × (TLB + Page Table + Mem)

Q. Consider a paging system with page table stored in memory.
(i) If a memory reference takes 50 nsec, how long does a paged memory reference take?
(ii) If we Add TLB & 75% of all Page Table reference are found in TLB, what will be effective mem A.T if finding an entry in TLB requires 2nsec.

⇒ (i) = 100nsec. (50+50 nsec)

(ii) $EMAT = \frac{3}{4} \times (2+50) + \frac{1}{4}(102)$

$= \frac{3}{4}(52) + \frac{1}{4}(102)$ ⇒ $\frac{78+51}{2}$ ⇒ $\frac{129}{2}$ ⇒ 64.5 nsec

Note - Unlike page table, there exist only one TLB in the system, so whenever the context switch occur, the entire content of TLB is flushed & deleted. TLB is then again updated with the currently running process.

- When a new process gets scheduled, the TLB will be empty initially so TLB misses are frequent.
- With every access from the page table, the TLB is updated & after some time, TLB Hit (↑) & TLB miss (↓).

Q. EAT = 160 nsec. MAT = 100 nsec. TLB Hit = 0.8

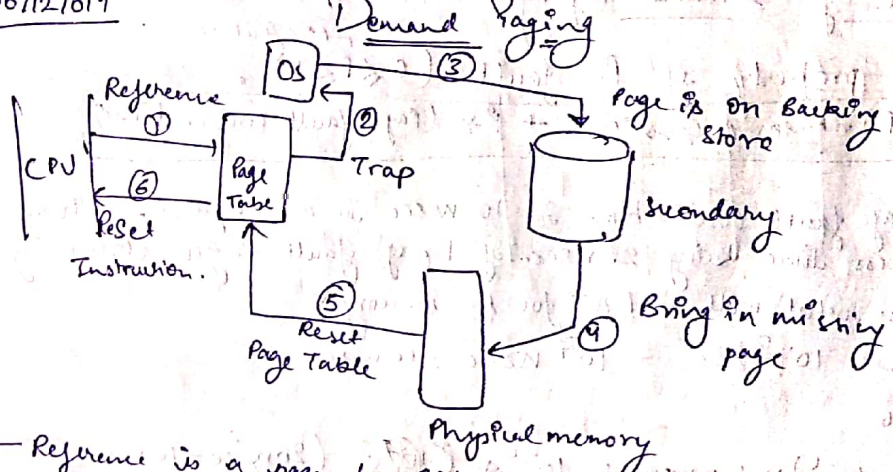
$160n = 0.8(TLB + 100n) + 0.2(TLB + 200n)$

$160n = 0.8\,TLB + 80n + 0.2\,TLB + 40n$

$TLB = 160n - 120n = 40$ nsec

06/12/2019

Demand Paging



- Reference is a page by CPU
- Check the entry of page in the page table along with valid/invalid bit.

1a - If the bit is set to valid
- Hit
- Calculate the physical address
- Fetch the Instruction Information.
- Execute

1b - If the bit is set to invalid
- Miss
- Check frame table for free frame
- Page Fault. 2m

2a. If free frame is available
- Allocate page to one of the free frame
- Update frame table
- Update page table and set bit to valid
- Follow 1a

(page requested by CPU, not found in physical memory)

2b. If free frame is not available
- Select victim page by applying page replacement algo.
- Send victim page to backing store
- Update page table and set corresponding bit to invalid
- Update frame table as one frame is free now
- follow 2a.

⇒ The process of loading the page into memory on demand, when ever page fault occurs, is known as demand paging.

* Performance of Demand Paging

Let P be the property of Page fault ($0 \leq P \leq 1$). ← page fault rate.

EAT = (1−P) × Mem Access Time + P * (Page fault service time)

Q. Let the page fault service time is 10 msec in a computer, with avg. memory access time being 20 nsec. If 1 page fault is generated, every 10^6 memory access, what will be EAT for the memory?

→ 1 msec = 10^{-3} sec = 10^7 nsec = 10 msec

P = $1/10^6$

EAT = $(1-10^{-6}) \times 20 + 10^{-6} * (20\text{nsec} + 10\text{msec})$ (20nsec + 10msec)

EAT = 20 + 10 = 30 nsec

- Page Replacement is a process of swapping out an existing page from the frame of main memory & replacing it with the required page. It occurs when all the frames of the main memory are already occupied.
- A good page replacement algo. minimizes the page fault.

MCU — MRU — random
FIFO — MFU — optimal
LRU — LRFU — LIFO

(1) FIFO – Oldest page in the memory will be replaced first.

Q. Consider the following reference string and calculate the no. of page fault (Hit Ration & Miss Ratio) on a physical memory of 3 frames.

7012, 0304, 2303, 2120, 1701

Belady's Anomaly: Frame ↑ Page Fault ↑ but in FIFO frame ↑ PF ↓ ⇒ Belady's

Page fault = 15
Hit Ratio = 5/20 = 25%
Miss Ratio = 15/20 = 75%

(2) LRU – 7012030423032120 1701

Page fault = 12
Hit Ratio = 8/20 Miss Ratio = 12/20

Note – In FIFO page algorithm, an unexpected result occurs. It is generally seen that by increasing the no. of frames in main memory, the no. of page fault must be reduced, but in certain reference of FIFO page replacement, this condition violates & no. of page fault increases with the increase of no. of frames with the memory ⇒ BELADY'S ANAMOLY

H.W
⇒ 1 2 3 4 1 2 5 1 2 3 4 5 (3, 4)

Q. The address sequence generated by tracing a particular program executing in a pure demand paging system with 100 records per page with one free main memory frame is recorded as follows. What will be the no. of page faults.

0100, 0200, 0430, 0499, 0510, 0530, 0560, 0120, 0220, 0240, 0260, 0320, 0370.

⇒ 1, 2, 4, 4, 5, 5, 5, 1, 2, 2, 2, 3, 3

✓ → page fault
✗ → no page fault

⇒ Total Page faults ⇒ 7

Hit Ratio → 6/13

Miss Ratio → 7/13

22/12/019 If exist then leave, if not then replace that

(3) Optimal - 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

(No queue)

R=7,0,1 R=2,0 R=3 R=0 R=4,2,3 R=0

P=3 P=4 P=5 P=6 P=7 P=8

Hierarchial Paging (Multilevel Paging)

- Inverted Page Table
- Threading

10/12/019 Segmentation (memory not divided (i.e. is round))

→ It is a memory management technique in which each process is divided into several segments of diff. sizes.

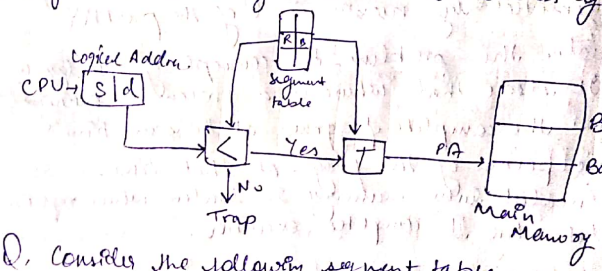
→ Segment is mem. management technique in which variable size chunks can be allocated to a process part and each chunk is called a segment

→ Each process maintains a table that stores information about each segment of the process. It has 3 columns.

(i) Segment No. ⇒ 1st column
(ii) Base / Starting Address of the segment ⇒ 2nd column
(iii) Limit / length of the segment ⇒ 3rd column

→ Segment table is stored as separate segment

→ Segment table base register stores the base address of the segment table



Q. Consider the following segment table.

Segment No.	Base	Length
0	1219	700
1	2300	14
2	90	100
3	1327	580
4	1952	96

→ Which of the following logical address will produce trap addressing : (0,430), (1,11) (2,100), (3,425), (4,95)

0 : 430 < 700 ⇒ correct address = 1219 + 430 (Base + offset)
= 1649 (P.A)

1 : 11 < 14 ⇒ Correct address = 2300 + 11
= 2311 (P.A)

2 : 100 ≮ 100 ⇒ Trap

3 : 425 < 580 ⇒ correct address = 1327 + 580 = 1907 (P.A)

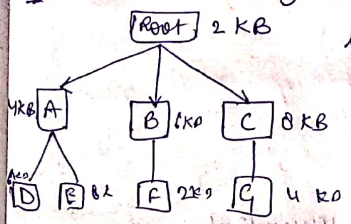
4 : 95 < 96 ⇒ correct address = 1952 + 95 = 2047 (P.A)

Overlays

- The main problem of fixed partitioning is the size of process has to be limited by maxm size of the partition.
- In order to solve this problem, the concept of overlay was introduced. It states that whenever a process is running, it will not use the complete program at the same time & it will use some portion of the prog. at that time.
- The portion needed into memory is loaded when required & is unloaded when it completes execution.
- Overlay is a technique to execute the program, of size bigger than main memory.

Eg. consider a program of 300 KB & an overlay manager of 20 KB. The program would be divided into 4 portions of 80, 80, 80 & 60 KB, since overlay manager will run all the time

Q. Consider an overlay tree. What will be the size of partition in physical memory to load & run this program?



RCG = 14
RBF = 10
RAD = 12
RAE = 14
} Maxm

∴ Ans = 14 KB

partition size ≥ 14 KB.

File Management System

⇒ A file is a named collection of related information that is recorded on secondary storage.
→ A file is a sequence of bits / bytes or records whose meaning is defined by its creator & its user.

* File Attributes

- Name, id

* File Extension

- Executable *exe, com, bin.
- batch bat, sh

File Operations

- A file is an abstract data type.

File Access Methods

(1) Sequential Access

- Used to read and write data consecutively
- Non-Random Access
- Very slow Access mechanism.
- 4 operation (Read next, Write next, rewind, forward)
- Long term durable (Utilised for long term storage & backup)
- Method used in magnetic tapes.

(2) Direct Access

- Reading and Writing data in any order
- locating a data item is easier.
- Random approach
- Faster access mechanism
- Used as primary method of storage
- Used for floppy disk, & CD, hard disk.

(3) Index Sequential

- This method is build upon sequential access
- An index is created for each file which contains pointers to various block,
- index is searched sequentially & its pointer is used to access the file directory.

* File Allocation Mechanism

- A single continous set of blocks is allocated to a file at the time of creation, thus this is pre-allocation strategy
- The file allocation table (FAT) needs to enter starting block & the length of file.

Disadvantage

- External fragmentation occur making it difficult to find the contigous block of memory of sufficient length.

(2) Linked List Allocation (Non-contigous)

- Allocation is on individual block basis
- Each block contains a pointer to the next block in the chain.
- File Table will have entry of starting block & length of the file.
- Any free block can be added to the chain, if required. & the block need not to be in continous.
- Expansion or increment in the file size is always possible, if free disk block is available.

Disadvantage - Overhead of maintaining pointer at any block

(3) Indexed Allocation

- solves the problem of both continous & linked list allocation
- A file allocation table contains a separate one level index for each file
- The index have one entry for each block allowed to file.
- This allocation supports both sequential & direct access.

Free-Space Management

- Disk space is limited
- Necessary to reuse the available space (free space) when files are deleted for new files.

(1) Bit-Vector

- To keep track of free disk space, free list is maintained by the system
- The free space list is implemented as bit map / bit vector
- If the block is free, bit = 1
  block is allocated, bit = 0

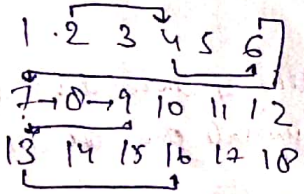
Ex. Assuming block 2, 4, 5, 6, 8 are free among 10 blocks, the bit vector will be → 0101110100

(2) Free-Block List

- Each block is assigned a number sequentially and the list of number of all free block is maintained in the reserved block of the disk.

(3) Linked-List Free Space

- Chained approach, that linked together, all the free disk block.



where, 2, 4, 6, 7, 8, 9, 13, 16, 12 are linked.

Grouping

- The address of (n) free blocks are stored in the first free block.

n=3 2, 4, 6, 7, 8, 9, 13, 16, 12