Lecture - 36

Segmentation

Divide in logically related partitions called segments.

By logically, we mean that stack of the process can be in a single segement. The complete body of a loop can be kept in a segment.

This reduces how many time we have to search for partitions like in paging.

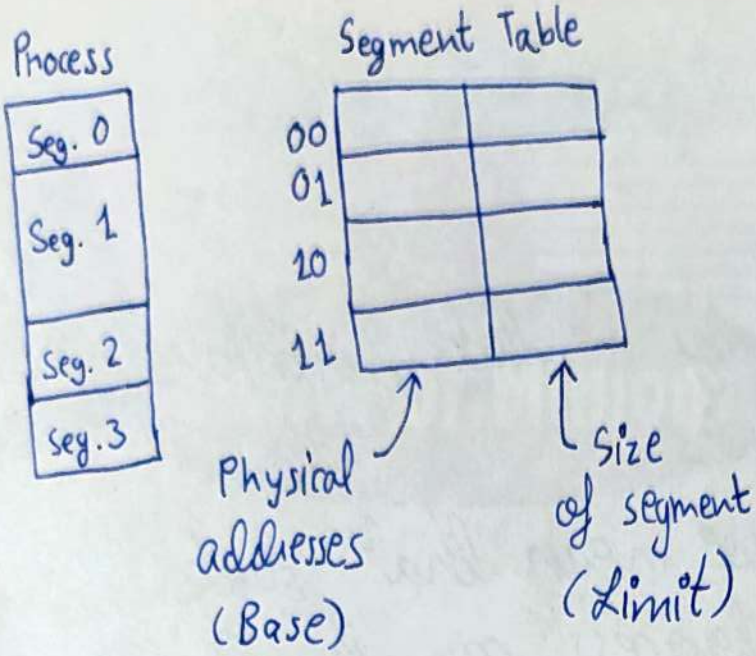
These segments can be of variable sizes

We will not partition our main memory.

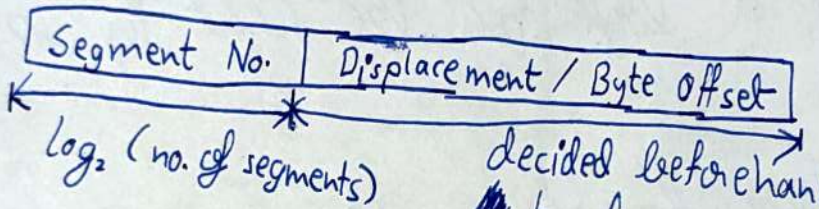
In segment table, we store,

Base ⇒ The physical address to the start of the segment.

Limit ⇒ Holds the size of the segment, used to achieve memory protection.



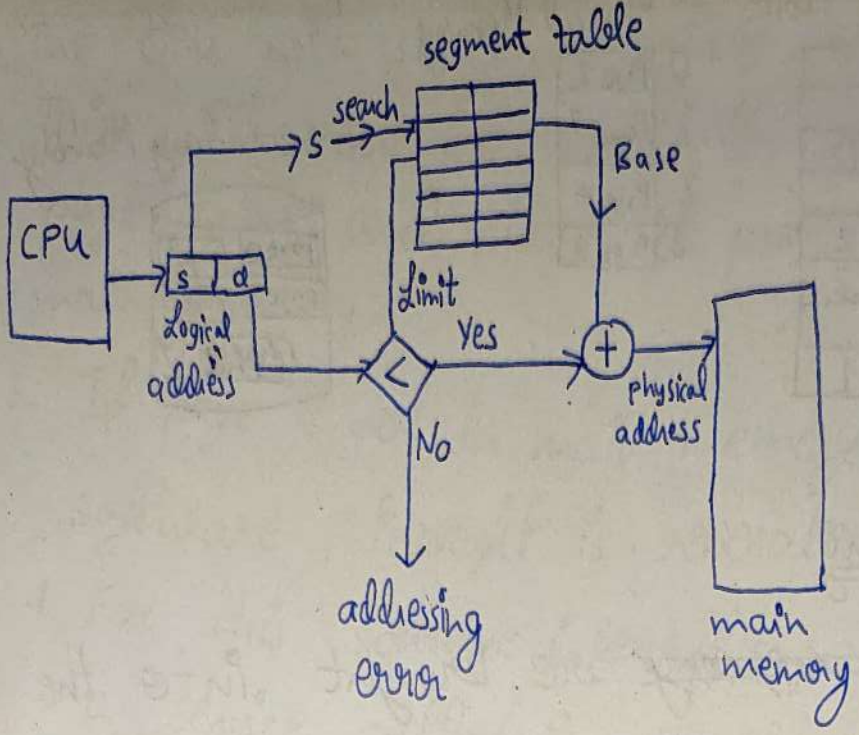
In segmentation, logical address,



To get physical address from logical address

Physical Address = Base (from segment table) + Displacement (from logical address)

Note: This is actual addition of the values rather than concatnation, We do concatnation in paging.

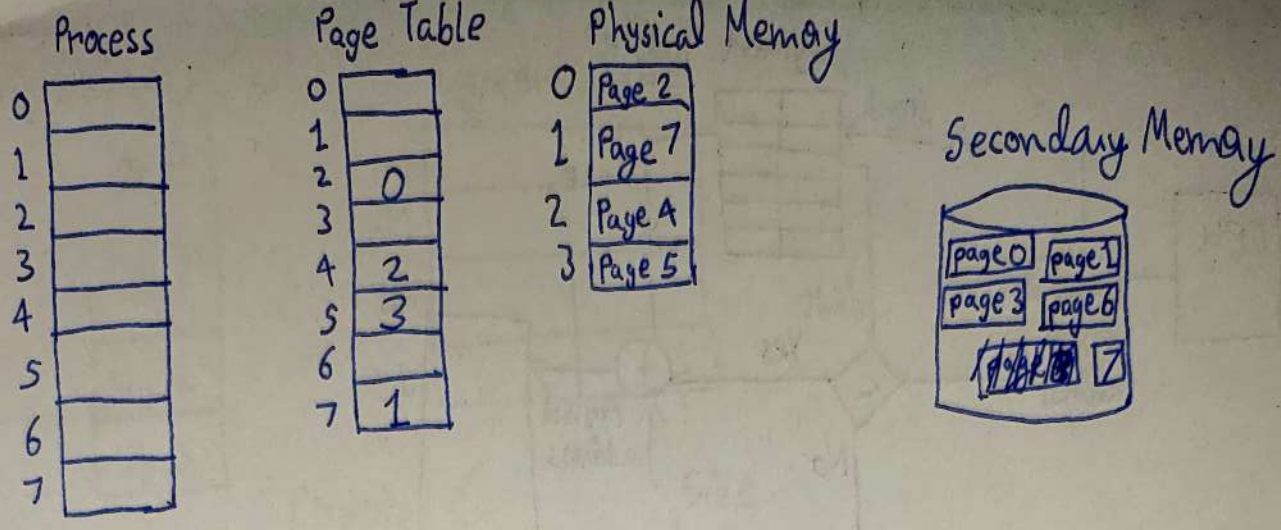


Lecture-37

Virtual Memory

A feature of O.S. It enables us to run larger process with smaller available memory.

In this concept, we store the pages which are needed currently in the main memory, rest are kept in the secondary memory.



Demand Paging

The pages ~~in memory~~ are brought into the main memory when CPU demands for them.

If CPU demands a page which is not available in main-memory, then it is known as page fault.

⇒ In case of a page fault, OS brings faulted page from sec-memory to main memory (replacing another page if needed) and updates the page table. This is called

⇒ ~~The page fault is detect~~ a page fault service.

1. The CPU will generate logical address.
2. This will now be used to search page table and check if page is in main memory.
3. If page is not in main memory, a software interrupt is generated
4. For this software interrupt, we do a service called Page Fault Service.

(Note: *DMA might be required for page fault service)

(DMA studied in COA I/O unit)

This type of paging is called demand paging

Demand Paging
- Pure demand paging
- Demand paging

i) Demand Paging:

Initially, before process execution starts, the most critical pages to start process execution are brought to the main memory.

ii) Pure demand Paging:

Initially, before process execution, no pages are brought into main memory, this causes page fault at the very start of process execution. Only the page table is created before process execution in pure demand paging.

Valid Bit

This is one of the extra bits which is stored in the page table for each entry.
It is used to detect page faults.

If valid bit is 1, then page is in main memory.
If valid bit is 0, then page is not in main memory and page fault is generated

Effective time (with page faults)

Effective memory access time = page fault ratio × (page fault service time)

+ (1 - page fault ratio) × (Effective memory access without taking page fault into consideration)

page fault ratio ⇒ The ratio of page faults per number of page table access.

Also called page fault rate.

Effective access time = TLB Hit × ($t_{mm} + t_{TLB}$) + TLB Miss × (TLB Miss time)

TLB Miss Time = Page fault ratio × (Page fault service time) + (1 − Page fault ratio) × ($2 \times t_{mm}$) + t_{TB}

Dirty Bit / Modified Bit

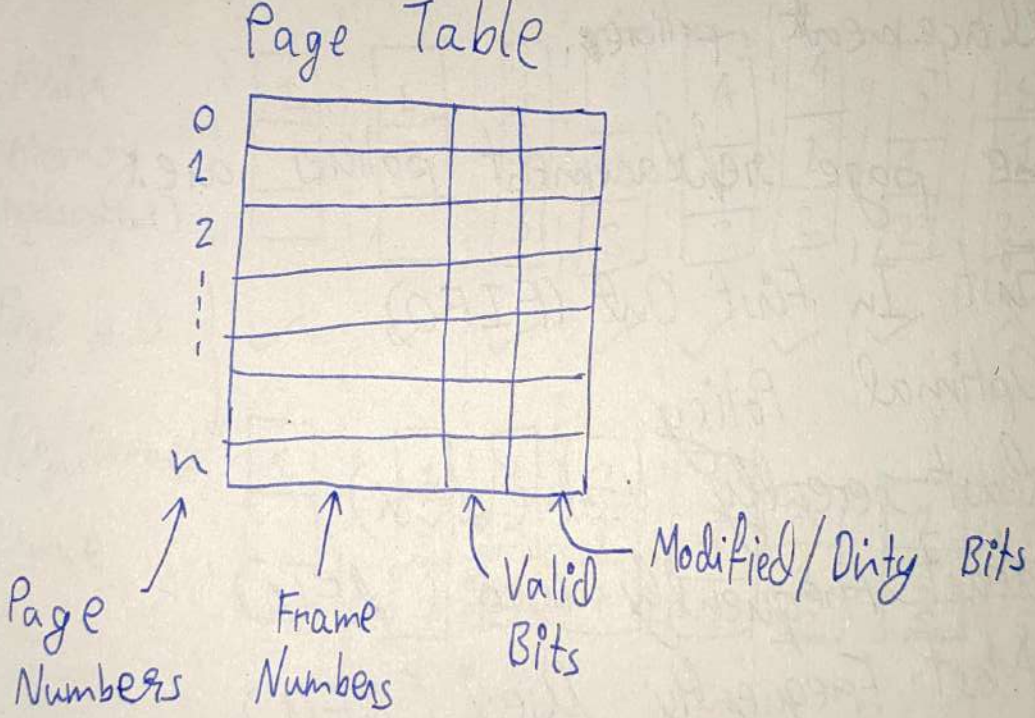
When we are replacing pages in virtual memory concept, there are two scenarios.

i) The page being replaced was not modified by the CPU (i.e, only reads were performed on the page) we can copy from secondary memory to main memory, requiring only a single copy operation.

ii) If the page being replaced was modified then we have to copy page from main to secondary, and then the new page from secondary to main requiring two copy operations.

Since copy operation from/to secondary memory and main memory is very costly, we want to reduce it.

Thus, we use modified bit/dirty bit to keep track of pages which were modified before replacing and doing the more efficient operation if possible



Lecture-38

Page Replacement

We have seen that in virtual memory concept, we replace pages in main memory with those needed from secondary memory.

Which page in main memory is decided by ~~policies~~, the implemented page replacement policies.

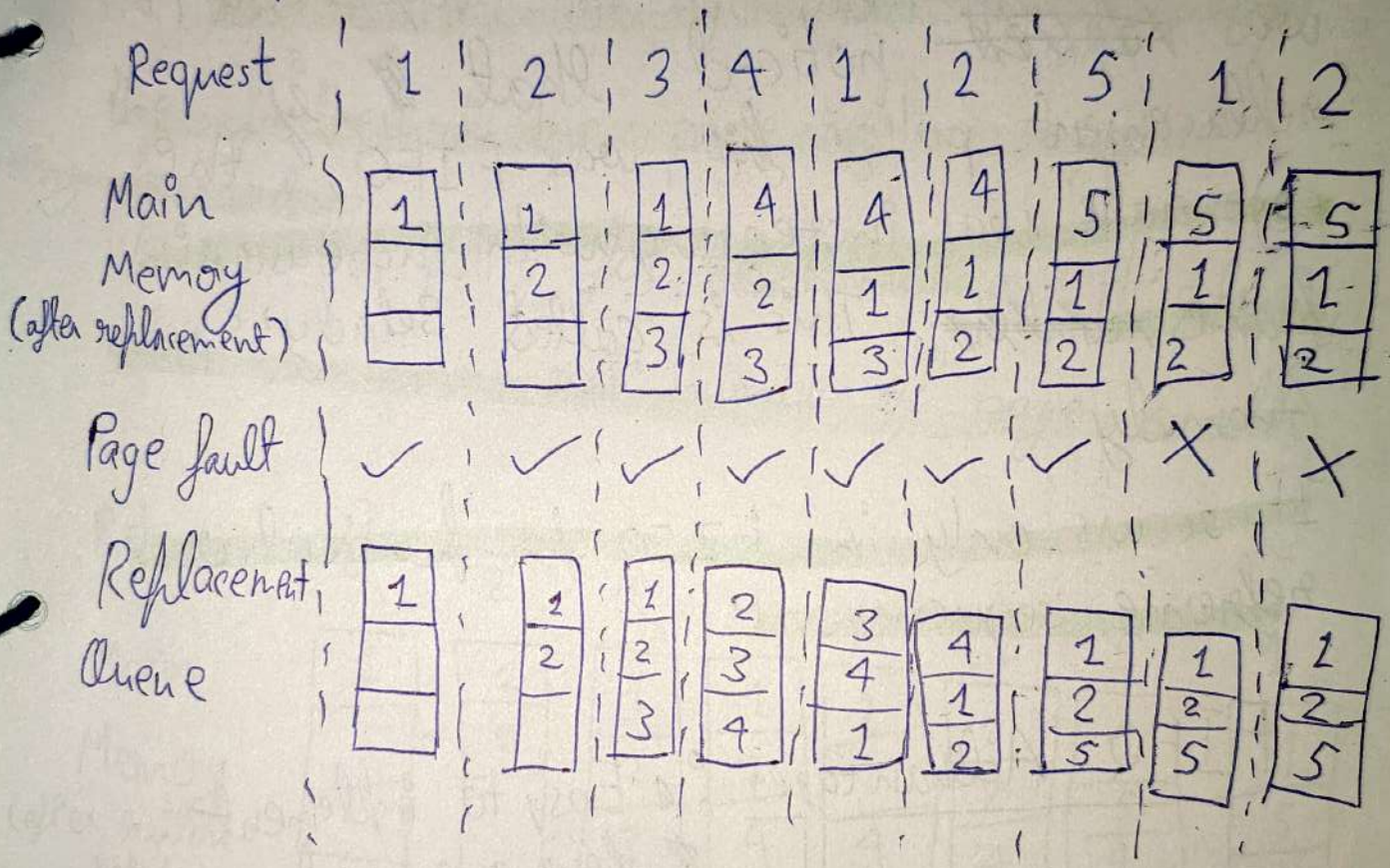
Some page replacement policies are:

i) First In First Out (FIFO)
ii) Optimal Policy
iii) Least recently Used. (LRU)
iv) Least Frequently Used (LFU)
v) Most Frequently Used (MFU)
vi) Last In First Out (LIFO)
vii) Second Chance

First In First Out

As the name suggests, we use a queue to replace pages. So the page that came into memory first is replaced first.

Example,



Number of requests = 9

Number of page faults = 7

Page fault rate = $\frac{7}{9}$

Belady's Anomaly

As we increase number of frames, generally page fault rate decreases. (as there are more frames, less replacement is needed).

But for some page refrence sequence, it was ~~Affected~~ noticed that if replacement policy ~~for~~ was FIFO, the page rate was increasing with increase in frames ~~numbers~~. This is called Belady's Anomaly.

It occurs only in FIFO for particular refrence sequences.

FIFO Advantages :
- Easy to implement
- Low overhead

FIFO Disadvantages :
- Poor performance
- Does not consider the frequency of use or last used time.
- Suffers from Belady's Anomally.

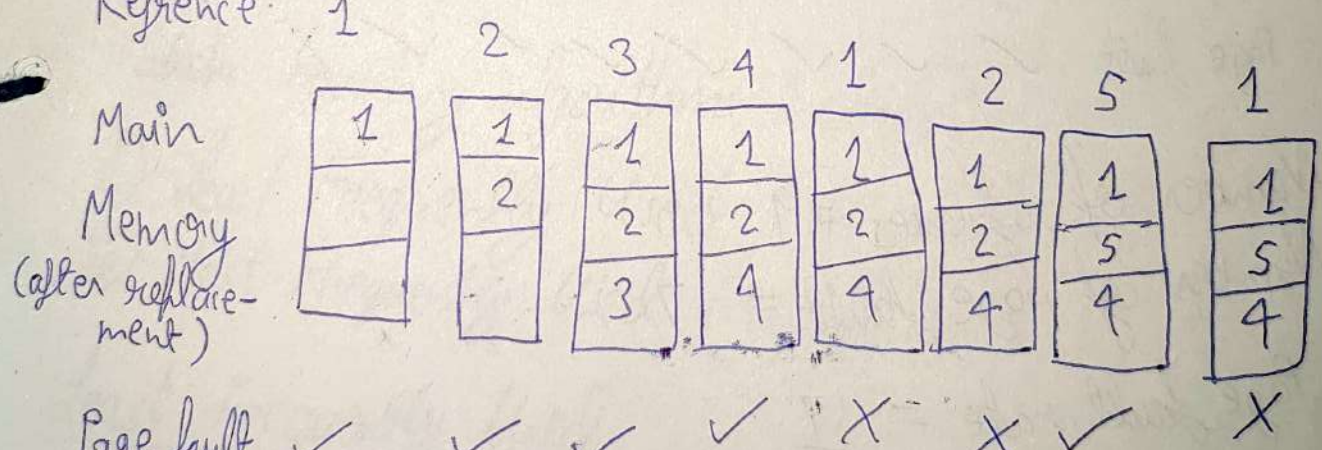
Optimal replacement policy

Assumes that we know all the page reference sequence (a theoratical replacement policy).

Replace page which is not (or never) going to be reffered in the near future.

i.e, we replace the page which is currently the farthest in the page refrence sequence going forward.

We use this policy for benchmark, as it always gives minimum number of page faults.



Number of page faults = 5

Number of refrences = 8

Page fault rate = $\frac{5}{8}$

Optimal policy advantages: ✱ Simple data structure used
✱ Highly efficient

Optimal policy disadvantages: ✱ requires knowledge of page reference sequence
✱ Comparison takes time so high overhead

Least Recently Used (LRU)

Replace the page which has not been referenced for the longest time.

Reference	1	2	3	4	1	2	5	1	2
Main Memory (after replacement)	1	1	1	4	4	4	5	5	5
		2	2	2	1	1	1	1	1
			3	3	3	2	2	2	2
Page fault	✓	✓	✓	✓	✓	✓	✓	X	X

Number of references = 9
Number of page faults = 7

Page fault rate = $\frac{7}{9}$

We replace the page which is farthest from the page fault causing reference (current reference) going backwards

LRU Advantages : * Efficient in practical scenario's for result.
* Doesn't suffer from Belady's Anomally.

LRU Disadvantages : * Complex implementation
* Expensive
* Requires hardware support.

Lecture-39

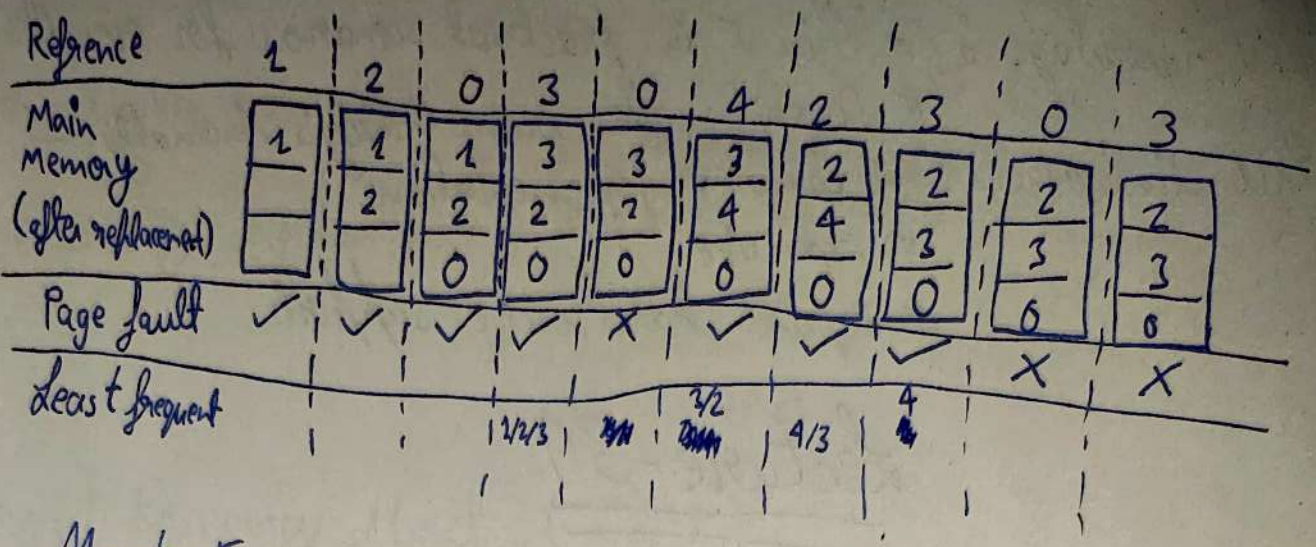
Counting Algorithms

Counting algorithms look at the number of occurances of a particular page in the reference sequence as a criterion for page replacement.

Such counting algorithms include:
i) Least Frequently Used (LFU)
ii) Most Frequently Used (MFU)

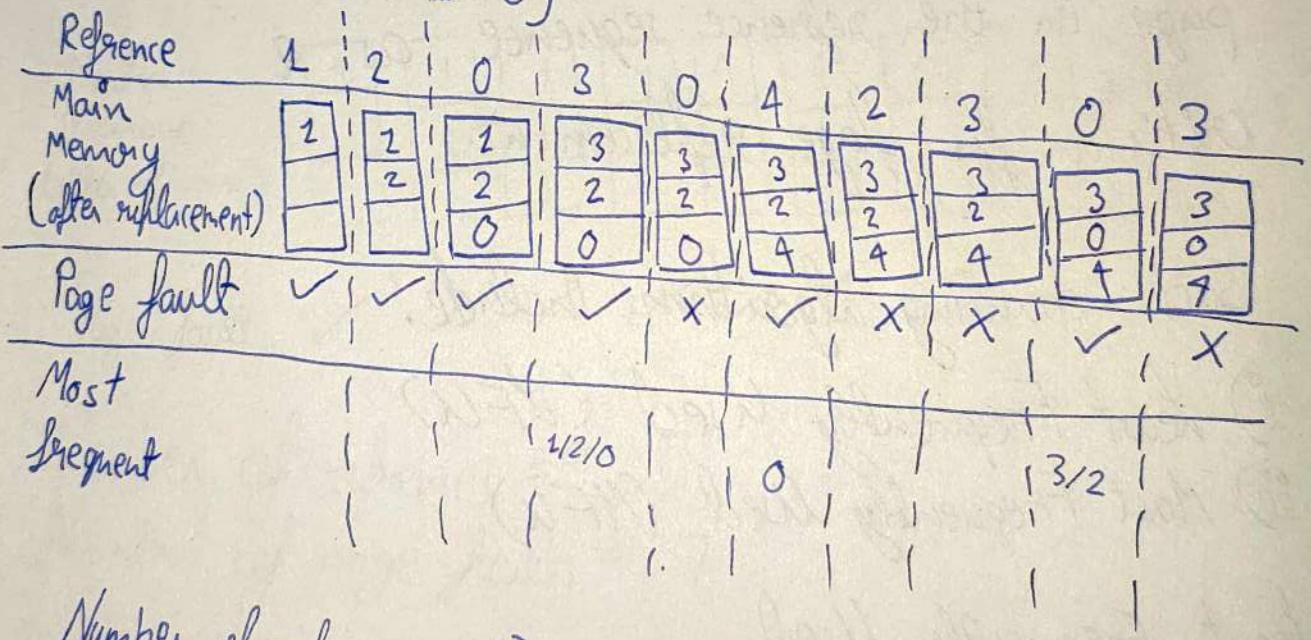
Least Frequently Used

Replace the page which was referenced least number of times till now in the reference sequence.
(Tie breaker : FIFO)



Most Frequently Used

Replace page which has been the most referenced in the reference sequence till now.
(Tie breaker : FIFO)

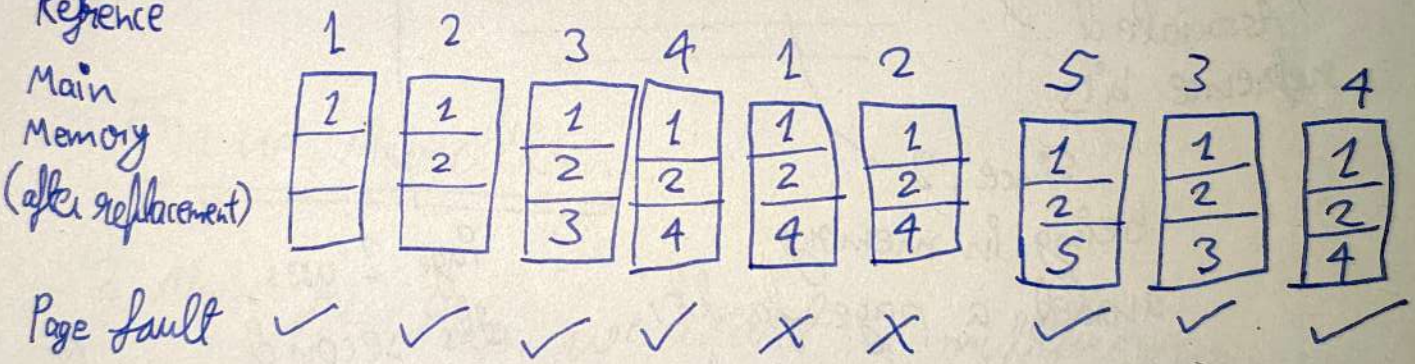


Number of references = 10
Number of page fault = 6
Page fault rate = $\frac{6}{10}$

Last In First Out

Replace the page which comes in physical memory last. ~~But this causes only a single page being replaced after the logical address space is filled.~~

But this causes the last frame in main memory which is used by the process to be the only one in which replacement occurs.

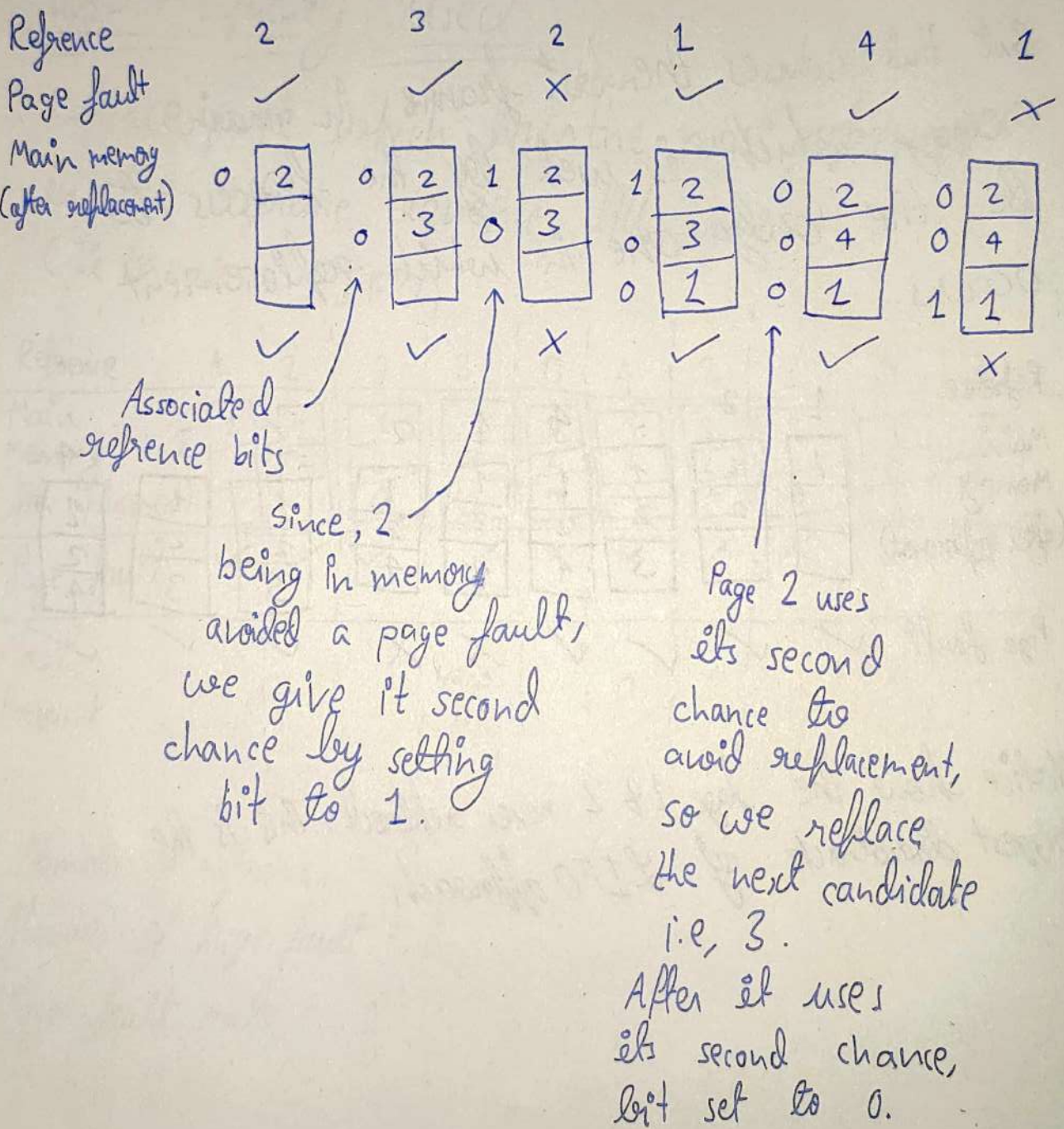


Notice how the page 1 & 2 never replaced. This is the biggest drawback of LIFO approach.

Second Chance

It is a variant of the FIFO method.
A second chance is given to pages, which were reffered by the CPU.

For this, we store an extra bit with each page in memory.



So in second chance:

1) When we bring new page to main memory, set it's refrence bit (R) to 0.
2) When a page helps avoid page fault, set its R=1
3) If FIFO selects page with R=1 for replacement, set its R=0 and look for some other candidate to replace.

Lecture-40

Frame Allocation

2 Main functions we want in multiprogramming:

i) Minimum number of frames that a process needs.
ii) Is page replacement global or local.

Minimum Number of Frames

Every process must have enough pages/frames in memory to complete an instruction

So,

minimum number of frames = no. of frames required for instruction of process

Frame allocation types

- Equal Allocation (Equal frames for process)
- Proportional Allocation (Frame allocated according to size of process)

For proportional allocation

$$\text{Number of allocated frames for process } i = \frac{S_i}{S} \times m$$

S_i = size of i^{th} process

~~S = size of the sum of all processes~~

S = sum of sizes of all processes

m = no. of frames in main memory.

Local Allocation & Global Allocation

The page of a process can only replace page of it's own process in local allocation.

In local allocation :

i) Local replacement requires that page being replaced belongs to the same process as the page coming from secondary memory.

ii) The number of frames belonging to a process will not change

iii) Thus processes can have controlled page fault rate.

In global allocation :

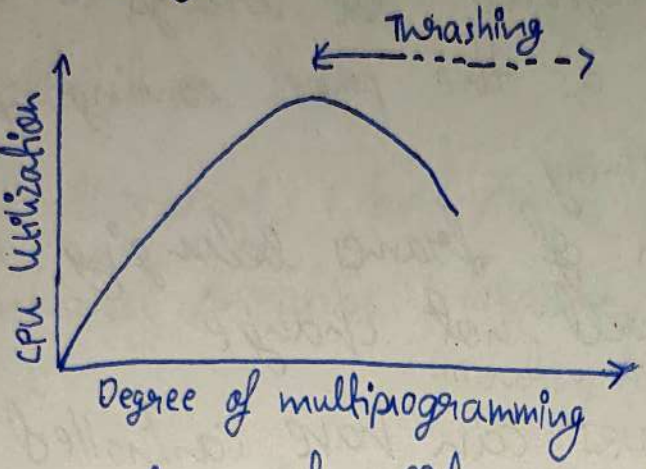
i) Page can replace any page from set of frames allocated for user processes

ii) High priority processes can increase their frame allocation at expense of lower priority process.

iii) Global allocation makes more efficient use of frames and has better throughput.

Thrashing

⇒ High level paging activity causes thrashing.



As we increase degree of multiprogramming, the CPU utilization increases.

But after a certain point, as there are too many processes, page fault rate and swapping rate also increase. This causes CPU utilization to start dropping. This drop is called thrashing.

As CPU utilization starts decreasing, long-term scheduler would try to increase CPU utilization by entering more processes into ready queue. This makes the CPU utilization even worse and the chain reaction continues.

How to handle Thrashing

1. Working Set Model (Based on locality model)
2. Page Fault Frequency

Working Set Model

The locality model says that when a process is running, at a given time, certain pages i.e, the locality of the process's logical space will be more frequently used. (This concept is also used in caching)

On basis of this, we use the working set model.

Suppose we have page reference sequence.

6 2 0 4 2 6 0 2 0, 3 1 3 1 0 3 1 1 0 3

In locality of pages 6,2,0,4

Locality of pages 3,1, 0

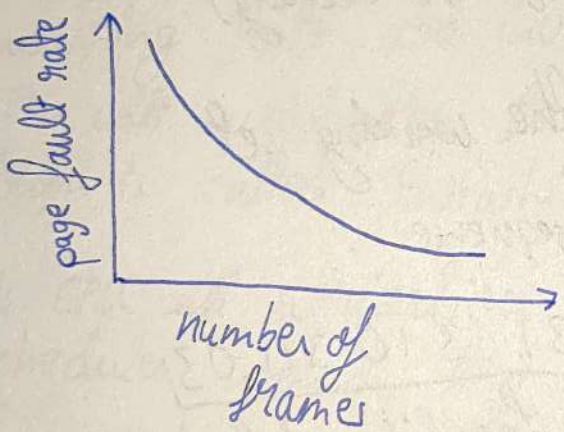
We create a working set to store the pages which are in the current locality and prevent them from being replaced.

⇒ If size of working set > locality pages, then pages of two different localities will be overlapping which is a wastage of main memory

⇒ If size of working set < locality pages, then page faults will occur causing a drop in CPU utilization.

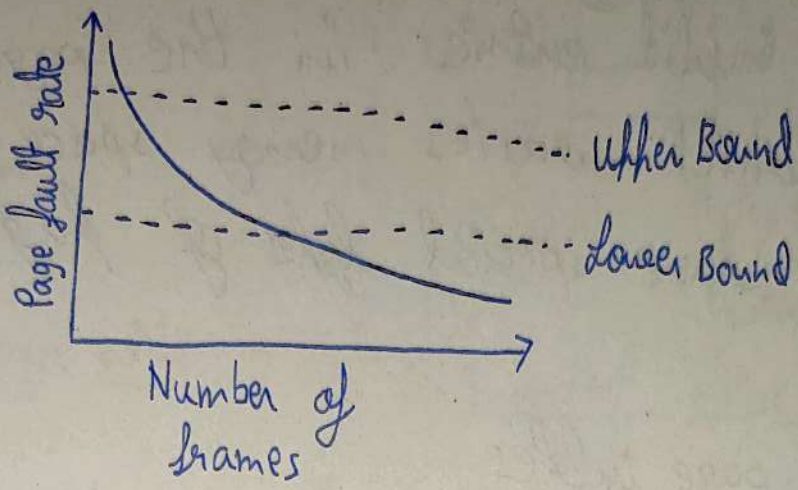
So we need to balance the size of working set.

Page fault frequency



Since number of frames ~~control the~~ affect the page fault rate, we can adjust no. of frames allocated to it, in order to affect page ~~danger~~ fault rate.

We choose an upper bound and lower bound for the page fault rate.



If page fault rate goes higher than the upper bound, we increase number of frames in order to keep it lower.

Similarly, if page fault rate goes below lower bound, we decrease the number of frames. to increase page fault rate.

Our goal is to keep page fault rate within our decided bounds.

Lecture - 41

In virtual memory concept, there can be multiple invalid entries in the page table which waste's memory space. So we have special types of page tables.

Types of page tables

1. Hierarchical Page Table (multilevel page table)
2. Inverted Page Table
3. Hashed Page Table

~~Invet~~

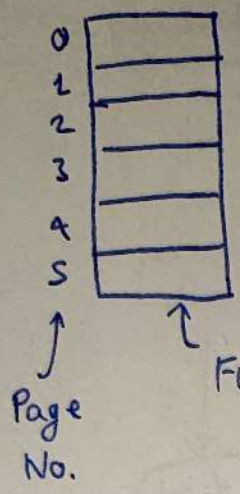
Inverted Page Table

In an inverted page table,

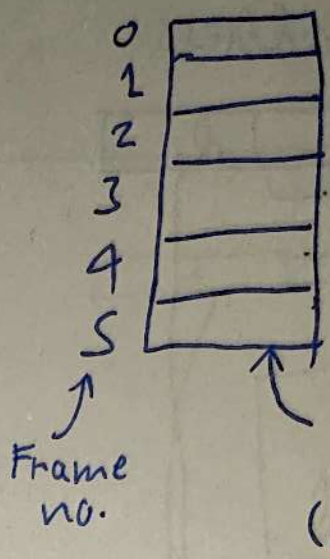
maximum no. of entries = No. of frames ~~allocated to the process~~ in memory

In the usual page table, we store frame numbers and page tables are represented by indexes.

In an inverted page table, this opposite.



Usual page table.



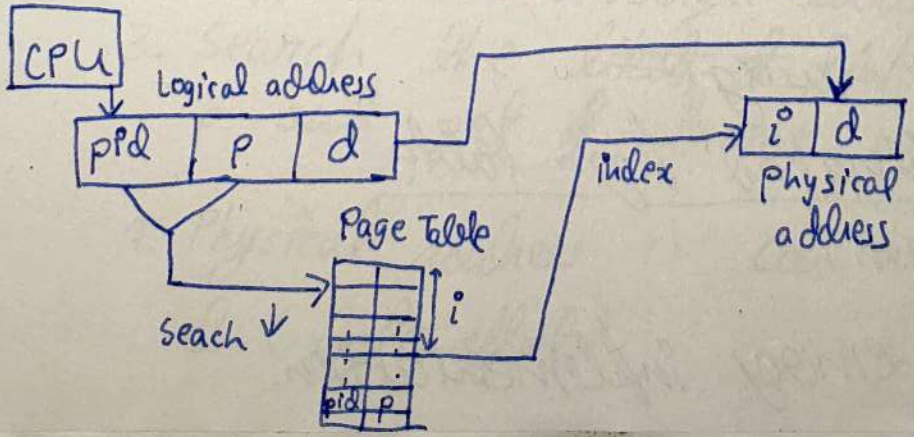
Inverted page table

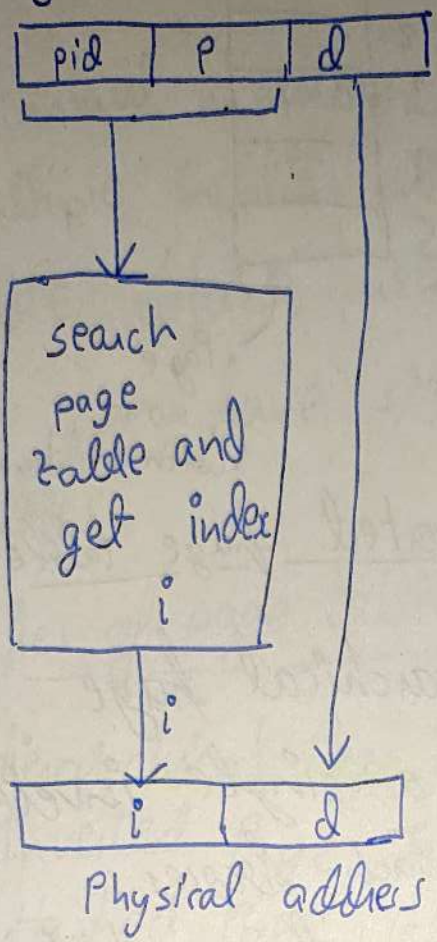
Note: Unlike usual & hierarchical page tables. There is only a single inverted page table in OS. (Each process does not have it's own page table)

Each entry in an inverted page table contains

1. Page Number } required
2. Process ID }
3. Control bits (Valid bits, dirty bits etc)
4. Chained pointer.

Logical to physical address





d = data offset / byte offset

p = page no.

i = index in page table

pid = process ID.

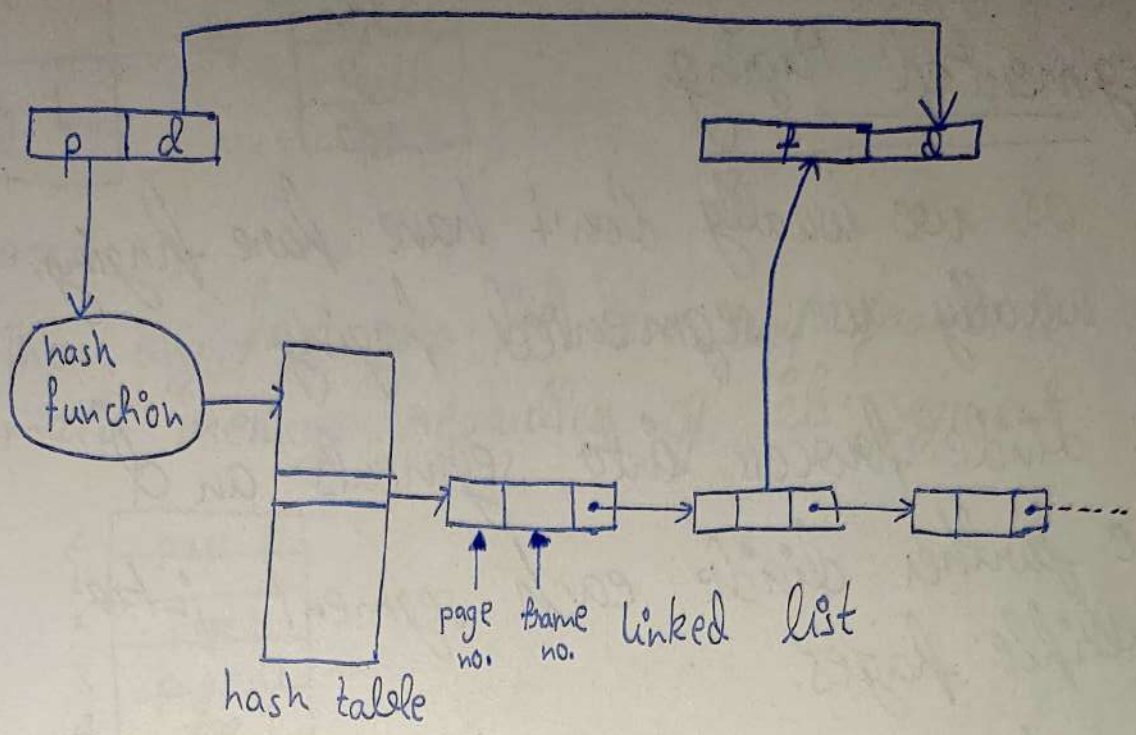
Advantage of inverted page table

Reduced memory usage.

Disadvantages of inverted page table

* Long lookup time
* Difficult shared memory implementation.

Hashed Page Table



In hashed page table, to convert a virtual address to physical address

1. Get page no. (p) and byte offset (d) from virtual address.
2. Pass p to a hash function to get linked list in hash table (multiple entries ~~for each~~ of pages can be mapped to a single location)
3. Search the ~~linked~~ linked list with p and get ~~the~~ frame number (f).
4. Physical address is concat of f and d (byte offset)

Lecture - 42

Segmented Paging

In OS, we usually don't have pure paging. We usually use segmented paging.

⇒ We divide process into segments and the further divide each segment into multiple pages.

⇒ We keep one page table per segment

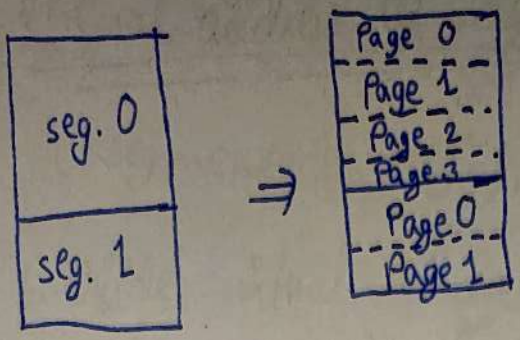
⇒ Segment table entry points to the page table of the segment.

⇒ Main memory has frames of size equal to page size.

⇒ There can be internal fragmentation when dividing segments.

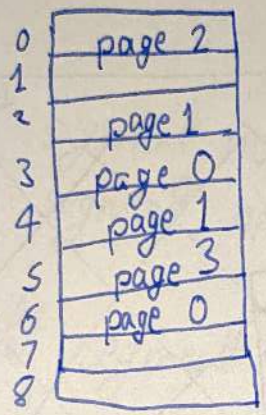
⇒ Any page can still be kept in any frame of the main memory.

Suppose,



Process

Then, any page can be in any frame in main memory regardless of its segment.



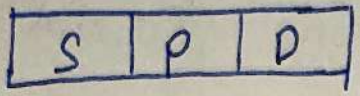
Main Memory

But as we can see above, we can't distinguish pages of segments this way.
For that, we keep page tables for each segment.

So each process has multiple segments and a single segment table. And ~~multiple~~ for each segment, there are multiple pages and a single page table.

Thus by logic, each process has multiple page tables.

Virtual Address



S ⇒ segment number.
P ⇒ page number of the segment.
d ⇒ byte offset of the page.

Virtual addres to Logical Address

(Segmented Paging)

1. Divide virtual address to segment number and segment offset.

Virtual Address

Segment No.	segment Offset
$\log_2$ (no. of segments)	$\log_2$ (maximum limit - 1) / Decided beforehand based on maximum limit

Let segment no. ⇒ s
segment offset ⇒ so

2. Let segment limit and page table base from s^{th} index of segment table.
3. If so < segment limit then valid entry and continue, else invalid entry and stop.
4. Divide segment offset to page number and byte offset.

Segment Offset

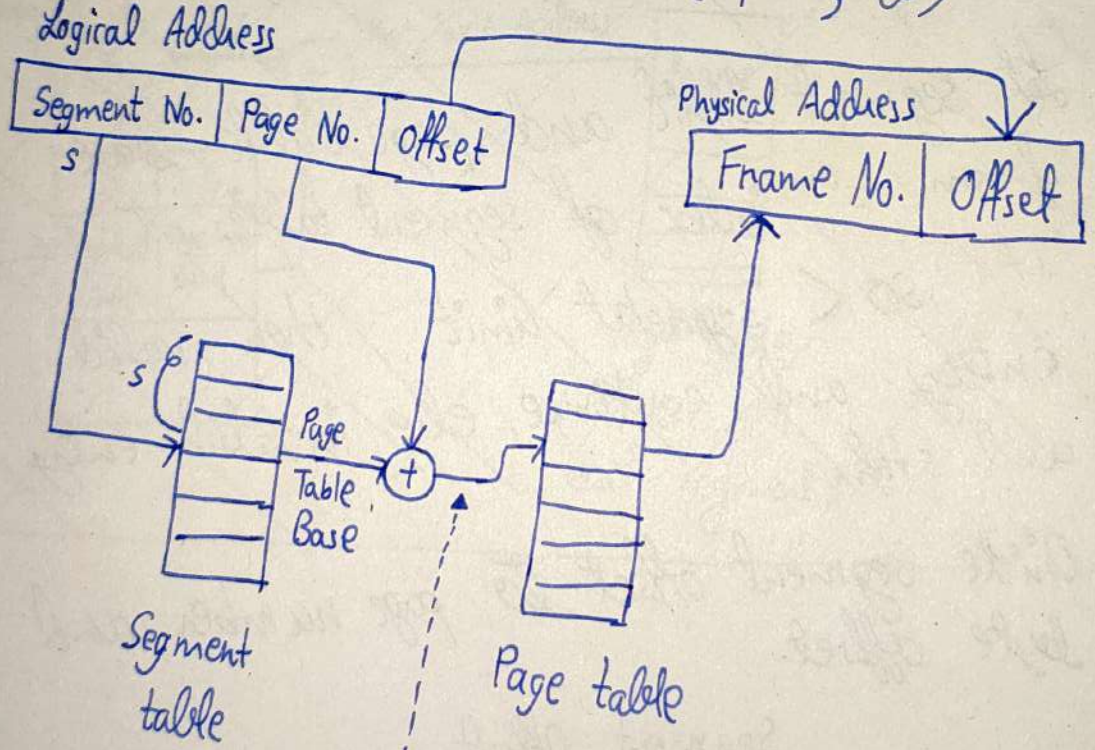
Page Number	Byte Offset
$\log_2$ (No. of pages)	$\log_2$ (frame size)

Let Page no. ⇒ p
Byte offset ⇒ d

5. Add p and Page Table Base (from segment table) (Note: Literally add, not concat).
to get Page Table Address.

6. Let frame number (f) from Page table ~~(at p th index of page table).~~
(at address, p + page table base)

7. Physical address = concat (f, d)



Dis Advantages of Segmented Paging

* Internal fragmentation still exists
* Extra hardware required
* Translation becomes more sequential increasing the memory access time.
* External fragmentation occurs because of varying sizes of page tables and segment tables.

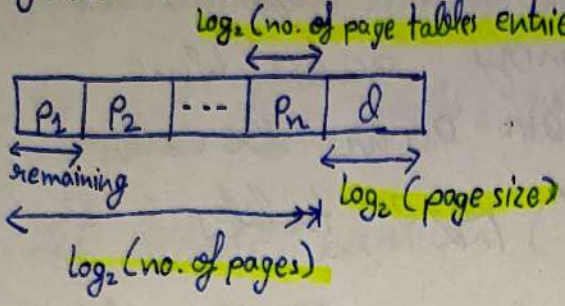
Advantages

* Page table size is reduced as pages are present only for data of segments.
* Gives programmer view along with advantages of paging.
* Reduces external fragmentation in comparison with segmentation.
* Since the entire segment need not be swapped out, swap operation into virtual memory becomes easier. (Reduces swapping)

Lecture - 43

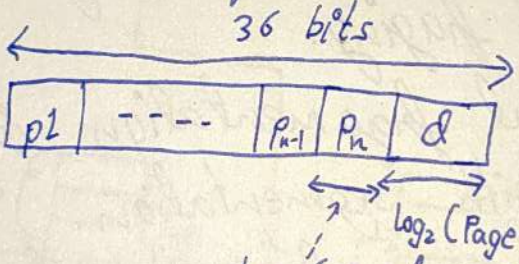
For multilevel paging

Logical Address



Eg, Consider a paged memory system with virtual addres of 36 bits and physical address of 28 bits. Page size is 2kB and page table entry size is 4 Bytes. How many levels of page table is required.

Sol)



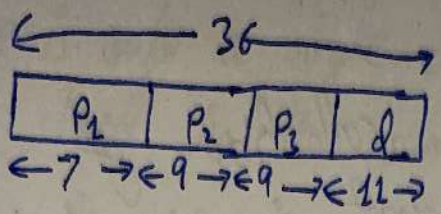
Page size = 2kB

Page size = 2^{11} B

$\log_2$(Page Size) = 11

no. of page table entries per page = $\frac{\text{page size}}{\text{entry size}} = \frac{2^{11}B}{4B} = 2^9 B$

So,
$\log_2$ (no. of entries per page) = 9



Hence, three level paging.

Optimal Page Size

It is the page size which will cause minimum overhead.

$$\text{Optimal Page Size} = \sqrt{2 \times E \times 2}$$

$\mathcal{L}$ ⇒ Logical address space (i.e process size)

E ⇒ Page Table Entry Size

So, $\text{optimal page size} = \sqrt{2\mathcal{L}E}$

Note: Lecture 43 also has questions on paging.

Lecture - 44

File

A file is a named collection of related information that is recorded on secondary storage.

File Directory

Collection of files.

File System

A module of OS which manages, controls and organizes files, directories and related structures.

Types of File Systems

- FAT32
- NTFS
- HFS+ (Used in apple stuff)
- Ext2/Ext3/Ext4
- Swap (Used for swapping)

File Directory Structure

1. Single-Level Directory

There is only a single directory and all files are stored in the same directory.

Limitations:

* All file names should be unique
* Cannot implement multiple users.

2. Two-level directory

We have a directory that contains multiple directories, each for a user. (Hence, named two level directory). Each user can put files in their own directory. But users cannot create more directories

3. Tree Structure Directory

The directory structure we use in modern operating systems.
There is a root directory in this type of structure from which we can access all files and directories and subdirectories.

Lecture - 45

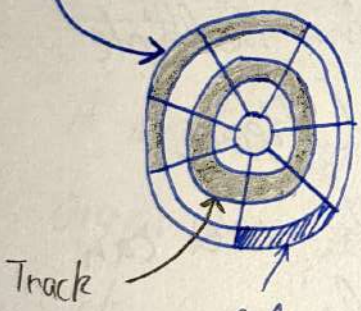
Magnetic Disk

(Also studied in COA subject)

We have multiple platters and we write on both surfaces.

Top View

Cluster (collection of consecutive sectors)



Sector
(Smallest unit of disk)
(Addressable unit of disk)

Disk Access Time = seek time + rotational latency + 1 sector transfe time.

seek time ⇒ time taken to move arm to correct track.

rotational latency ⇒ time taken to rotate platter to correct position

If rotational latency not given to us, we use the average rotational latency.

avg. rotational latency = $\frac{1 \text{ disk rotation time}}{2}$

1 sector transfer time = $\frac{1 \text{ disk rotation time}}{\text{no. of sectors per track}}$

(Check COA for more info)

Lecture - 46

Disk Partitioning

- Low-level (Tracks and sectors; done by manufacturer) (Physical partitioning)
- High-level (Done by user) (Logical partitioning)

Logical Formatting

2 types of partitions

- Primary (Bootable partition)
- Extended (Non-bootable partion)

Every OS requires a primary partition.

Only a single OS can be kept in a single primary partition

Disk Blocks

Created by the operating system. Disk block is a logical representation of smallest unit of disk.

1 block is usually of 1 or 2 disk sectors.

A file smaller than our block size will still occupy a complete block (files occupy whole blocks), thus there is internal fragmentation.

This is why size on disk is usually larger than actual size.

Free Space Managment

The manggement of the free space / free blocks on system is done mainly in two ways.

1. Free list
2. Bitmap method

Free List

We maintain a linked list of all free blocks.

~~If a block is used~~

We get a block from list when we want to allocate.

When a block is freed, we add it to the list.

Bitmap Method

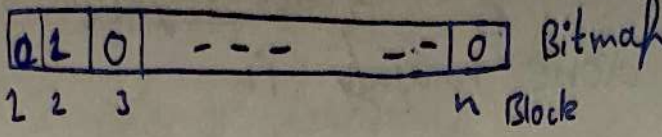
For each block we store a single bit

Bit 0 ⇒ block is free

Bit 1 ⇒ block is occupied

Thus we will for n-blocks, keep a n-bit data

called Bitmap which will store if block is used.



Free list vs Bitmap

* No searching in free list, but in Bitmap we first ~~seach~~ search for empty block (bit 0)
* Free list is faster in allocating a free block.
* Free list is variable in size, whereas bitmap size is constant.

Lecture-47

File Allocation Methods

Ways to select free blocks to store new file.

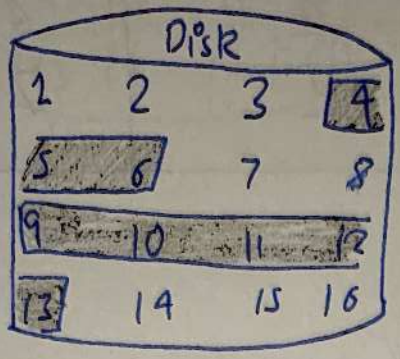
1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

We use file allocation table to keep track of our allocations.

acrual size.

Contiguous Allocation

We keep a File allocation Table



File allocation table

File name	Start Block No.	no. of block
abc.doc	4	3
xyz.pdf	9	5

Files are stored in contiguous blocks.

⇒ This method has both internal and external fragmentation.

⇒ Increase in file size is hard to accomadate. Thus it is inflexible

⇒ We can have sequential as well as random/ direct access.

Linked Allocation

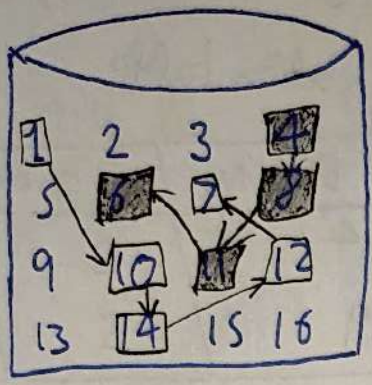
Uses free list method.

Files are also stored in linked list form.

abc.doc : 4 → 8 → 11 → 6

xyz.pdf : 1 → 10 → 14 → 12 → 7

In allocation table, we will keep start block and end block.



File allocation table

File name	Start Block no.	Last Block
abc.doc	4	6
xyz.pdf	1	7

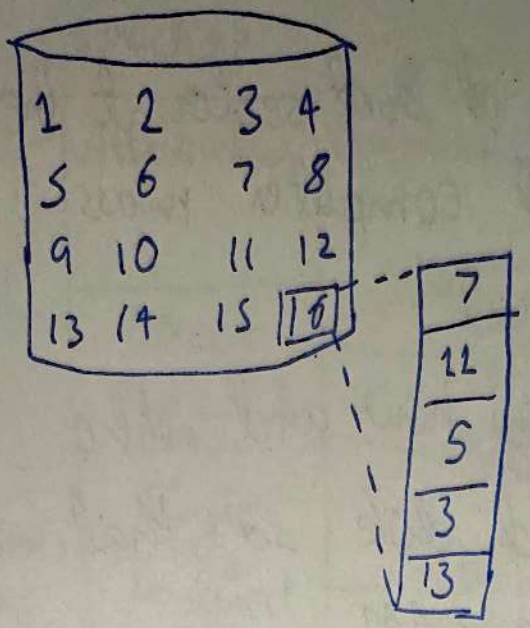
⇒ Only internal fragmentation is present.
⇒ It is flexible, files can increase size easily
⇒ Only sequential access

Indexed Allocation

We keep a seperate block with each file to store it's block locations, which we can then access. So in file allocation table we only need to store the index block location.

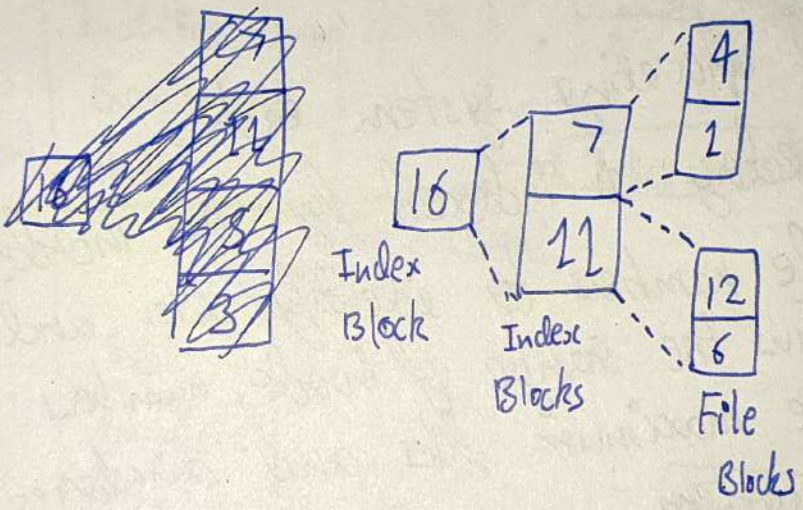
Similar to multilevel paging, if a single index block is too small for our index, we can have multilevel indexing.

File name	Index Block
xyz.pdf	16



index Block

Similar to paging, we can have multiple levels.



⇒ Only internal fragmentation

⇒ Flexible and can easily change file sizes

⇒ Both sequential & direct access allowed.

Master Boot Record (MBR)

A MBR is a special type of boot sector at the very beginning of partitioned computer mass storage device.

Contains information regarding how and where the OS is located in hard disk so that it can be booted in the RAM.

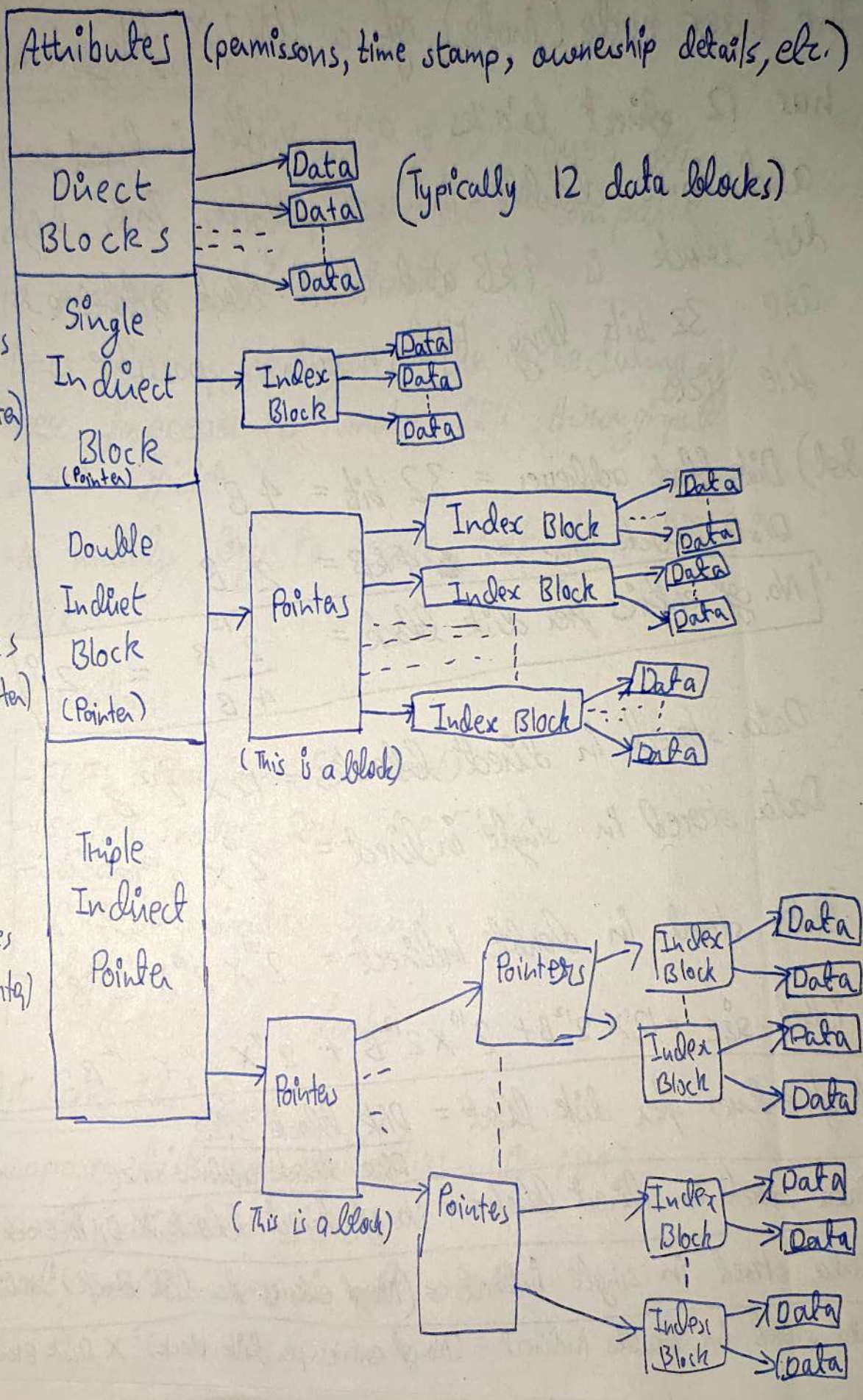
Unix I-Node Structure

In unix based operating system each file and directory is indexed by an inode. We use inode numbers to identify files and directories. Thus the range of inode numbers will determine maximum files and directories in the file system.

For each inode number we have a inode structure which store the attributes of the file/directory.

Inode structure stores attributes and disk block locations.

Inode structure



Example,

The index node (inode) of a UNIX filesystem has 12 direct blocks, one single indirect, and one double-indirect pointer. The disk block is 4kB and disk block addresses are 32 bits long. Find maximum possible file size.

Sol) Disk block addresses = 32 bits = 4 B

Disk block size = 4kB = 2^{12} B

No. of entries per disk block = $\frac{2^{12} B}{4 B} = 2^{10}$

Data stored in direct blocks = 12×2^{12} B

Data stored in single indirect = $2^{10} \times 2^{12}$ B

Data stored in double indirect = $2^{10} \times 2^{10} \times 2^{12}$ B

Total size = $12 \times 2^{12} B + 2^{10} \times 2^{12} B + 2^{10} \times 2^{10} \times 2^{12} B$

No of entries per disk block = $\frac{\text{Disk block size}}{\text{Disk block address size}}$

Data stored in direct blocks = No. of direct block × Disk Block Size

Data stored in single indirect = (No. of entries per disk block)^2 × Disk Block Size

Data stored in double indirect = (No. of entries per disk block)^2 × Disk Block Size

Lecture- 48

Disk Scheduling

Processes which have I/O request for disk are usually very slow compared to processes which don't.

Thus disk scheding is the scheduling of these processes to make CPU throughput more efficiet.

We mainly try to save seek time in disk.

Algorithms

- FCFS (Fist Come Fist Serve)
- SSTF (Shortest Seek Time Fist)
- Scan
- C-Scan (Cicular Scan)
- Look
- C-Look (Cicular-Look)

Fist Come Fist Serve

Suppose disk has cylinders (0-199)
no. of cylinders = 200
Read/Write arm is at 50

Suppose requests are
cylinder : 72, 160, 33, 130, 14, 6, 180

In FCFS, we serve requests in the sequence they come so,

50 → 72 → 160 → 33 → 130 → 14 → 6 → 180

We can subtract to get number of head movements by subtracting (bigger position from smaller)

50 $\xrightarrow{22}$ 72 $\xrightarrow{88}$ 160 $\xrightarrow{127}$ 33 $\xrightarrow{97}$ 130 $\xrightarrow{116}$ 14 $\xrightarrow{8}$ 6 $\xrightarrow{174}$ 180

Total head movements = 22+88+127+97+116+8+174
= 632

So if we know the time per head movement, we can get seek time.

Seek Time = (Total head movements × time per head movement)
+ extra delay
↑ If mentioned

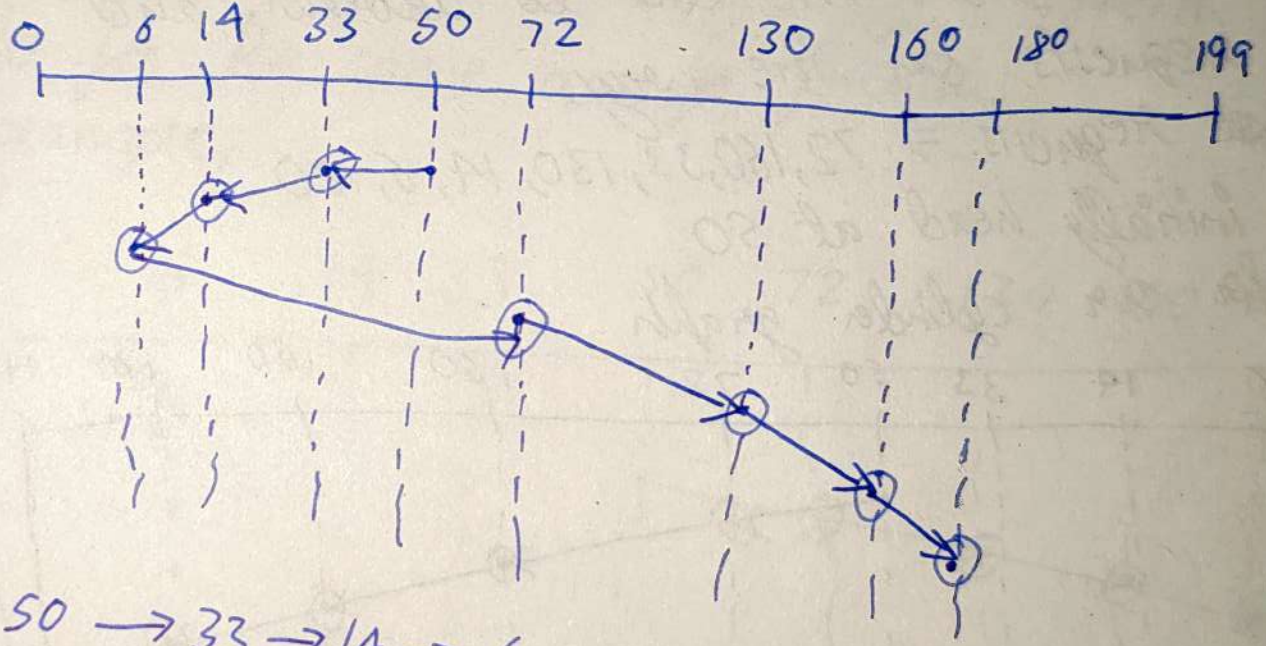
Advantages of FCFS
* Every request gets fair chance
* No indefinite postponement.

Disadvantages of FCFS
* Does not try to optimize seek time
* May not provide best possible service

Shortest Seek Time First (SSTF)

We will select the closest cylinder request to our current head position.

Suppose cylinder request, 72, 160, 33, 130, 14, 6, 180
Head is at 50 at start,
We will find the closest cylinder
Suppose given below is our cylinder tracks. (circles are completed requests)



So, $50 \xrightarrow{17} 33 \xrightarrow{19} 14 \xrightarrow{8} 6 \xrightarrow{66} 72 \xrightarrow{58} 130 \xrightarrow{30} 160 \xrightarrow{20} 180$

Total head movements: $17+19+8+66+58+30+20$

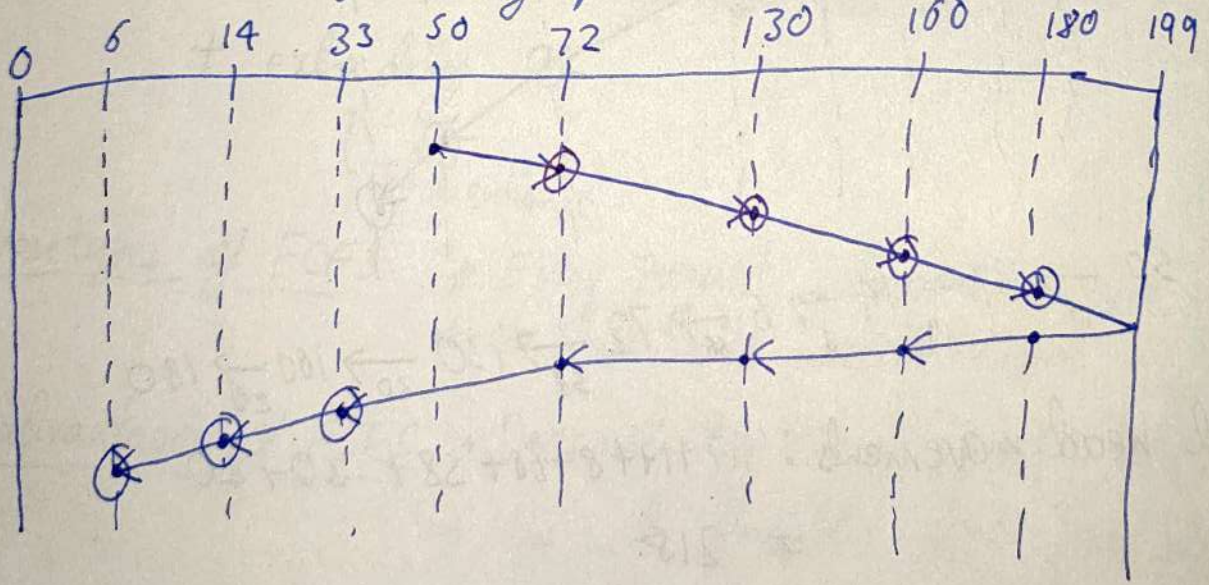
$= 218$

Advantages of SSTF : ✱ Average response time decreased
✱ Throughput increases

Disadvantages of SSTF : ✱ Overhead of calculating seek time in advance.
✱ Can cause starvation if processes in the same cylinder region keep getting scheduled
✱ Favours only some requests, hence biased.

Scan (Elevator)

We scan from one end to another, and do requests on the way.
Suppose requests. ⇒ 72, 160, 33, 130, 14, 6, 180
AND initially head at 50.
So for our cylinder graph



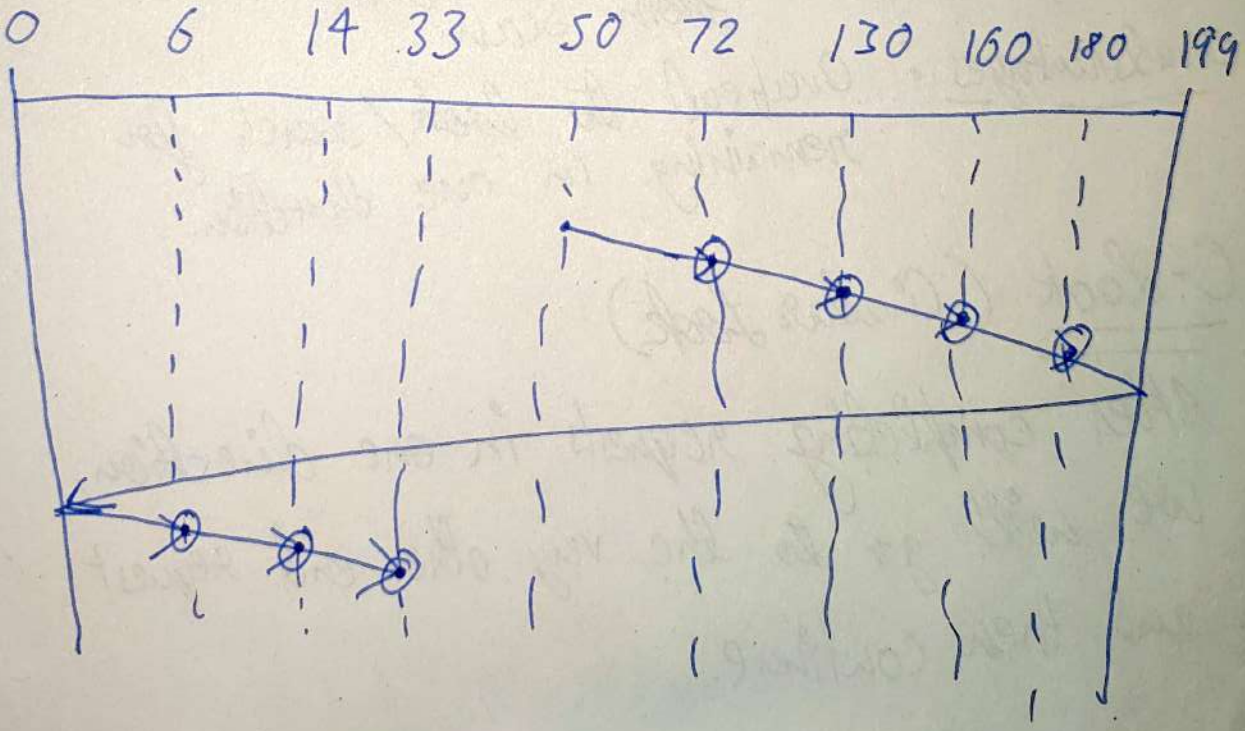
50 → 72 → 130 → 160 → 180 → 199 → 33 → 14 → 6

Advantages of Scan
* High throughput
* Low variance on response time
* Average response time.

Disadvantages of Scan
* Long waiting time for requests of locations just visited.

C-Scan

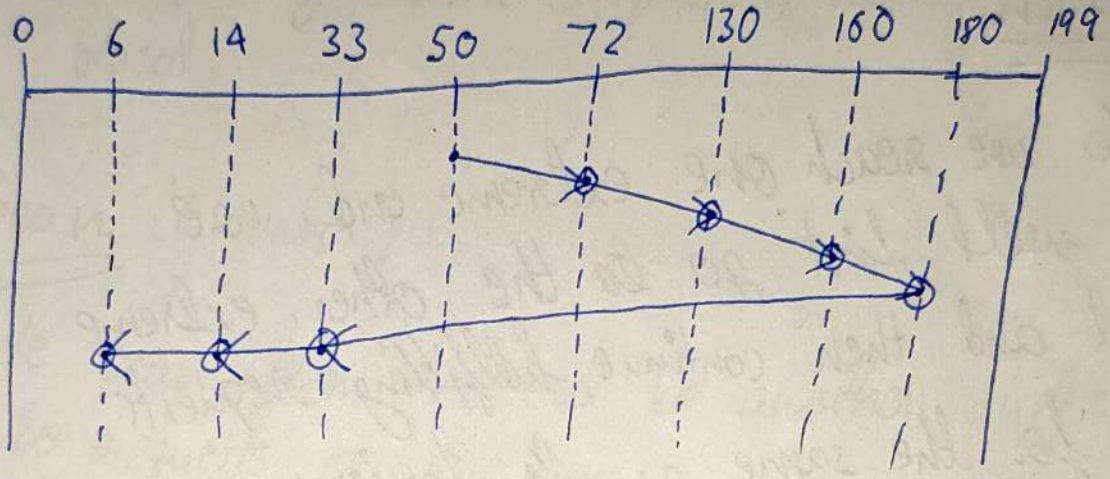
After we reach one extreme end, we will first go to the other extreme end and then continue fullfilling requests. So for the same as previous example,



This makes response times more consistent.

Look

It is a small improvement to Scan. Rather than going to the extreme end, we turn back when requests of one direction are done. So for our example.

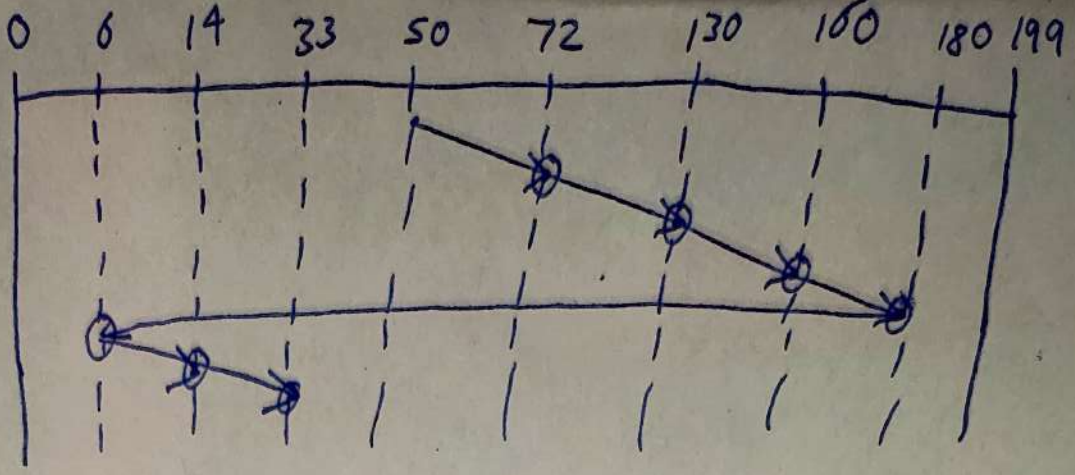


Advantages of Look: Improvement in performance from Scan

Disadvantages: Overhead to look/search for remaining in one direction.

C-Look (Circular Look)

After completing requests in one direction, we will go to the very other end request and then continue.



Head movements = $(180-50)+(180-6)+(33-6)$

Note: If in above type of graph there is a straigh line as in case of LOOK, C-Look, Scan and C-Scan. We can reduce calculation by subtracting points of straight lines as shown above (By hoits we mean cylinder value at those points.