25/3/2018
class-26

→ Management of main memory
→ Management of I/O devices
→ Management of storage devices

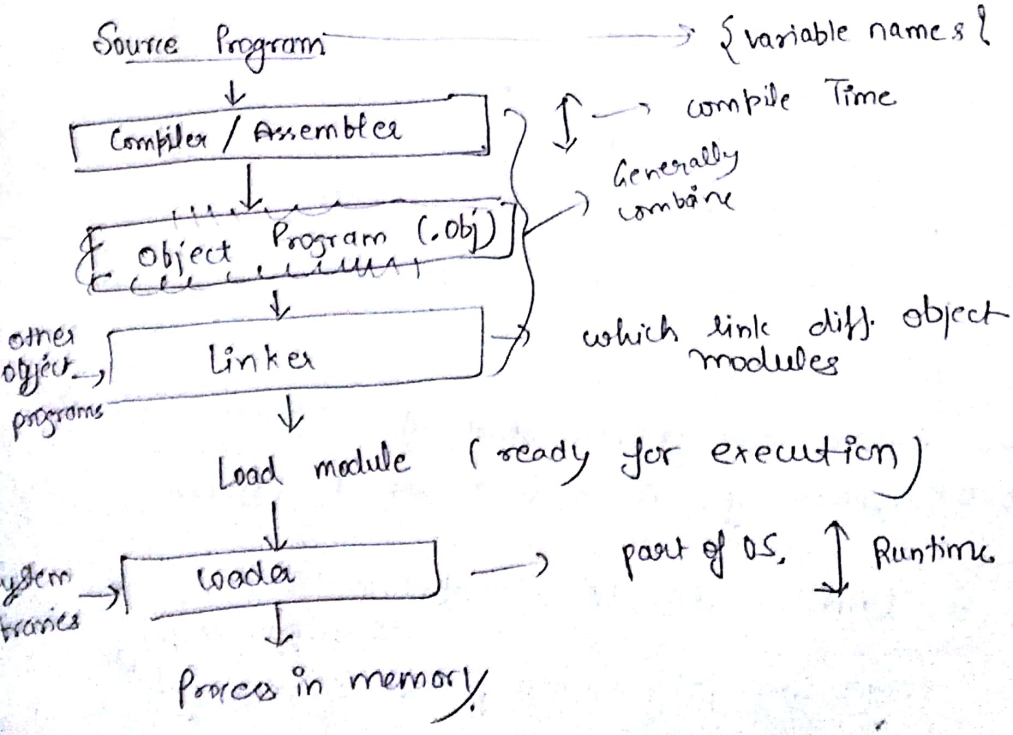
MEMORY MANAGEMENT

Primary or Main Memory
↳ the memory which is processor accessing directly.
RAM and ROM.

Management of RAM

Why?
To execute the process, we have to load into RAM.

→ How a ~~process~~ program becomes a process?



Any activity during compilation is in compile time.

Ex example -

```
c = (2 * 3.14) * r;

c = 6.28 * r;   ---> constant folding (not in syllabus.)
```

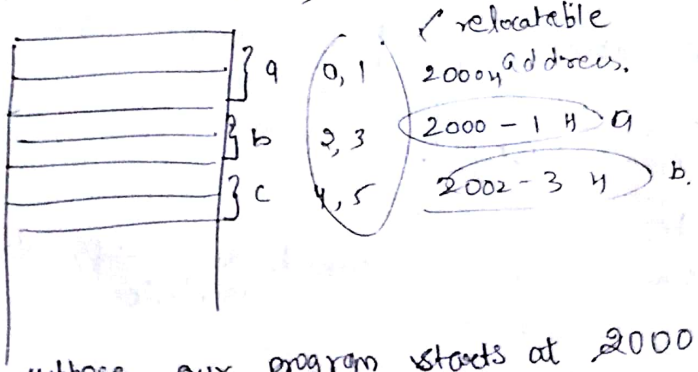
When you are executing a process, then any activity at that time, is called run time activity.

→ (Variable name) is a logical name for some memory location. → In source program

after that no variable names.
(compiler removes all variable names)

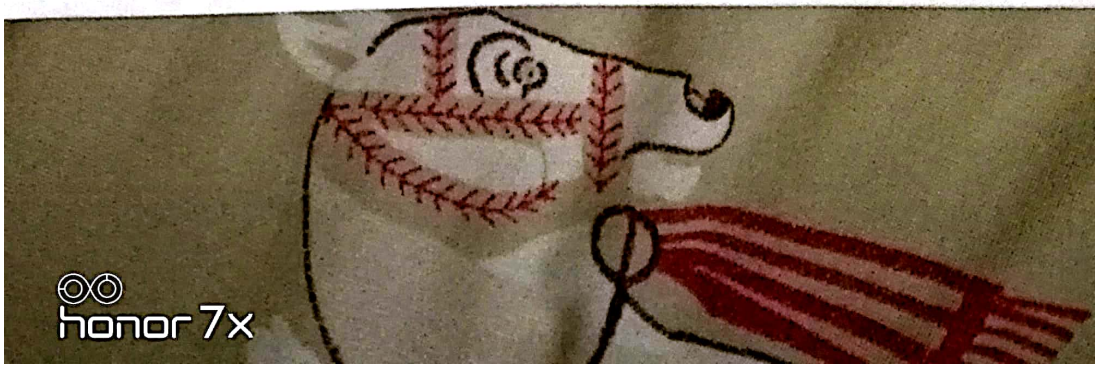
At Load module → Relocatable addresses.

Ex- int a, b, c;



Now suppose our program starts at 2000 H, then absolute address of a will be 2000H - 2001H

LOA 2000H ← absolute address programming comes into play when process is loaded.

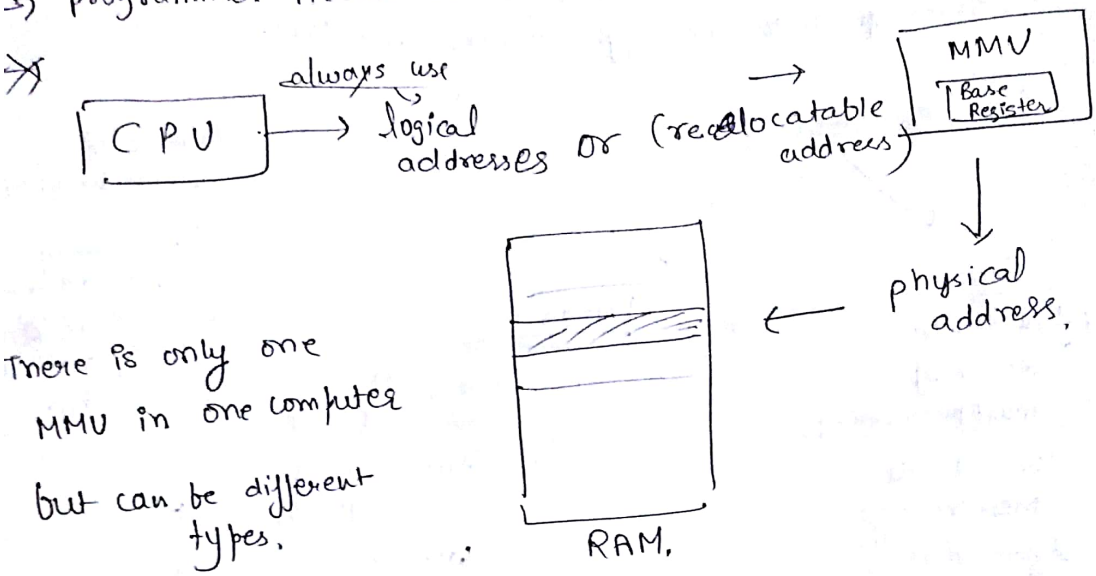


If we use absolute addressing,

→ implementation of multiprogramming and multitasking will become difficult.

→ we have to keep track of address (available locations oneself)

→ programmer needs to know the size of memory.

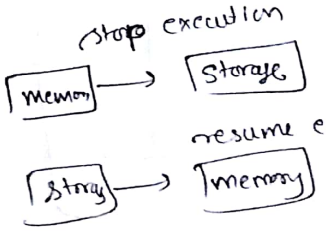


There is only one MMU in one computer but can be different types.

Swapping.

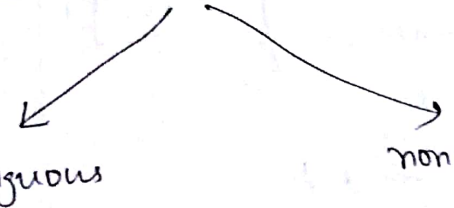
Swap Out

Swap In.



If we combine swapping with priority ⇒ then called roll out, roll in

Memory Management Schemes.



Contiguous

non-contiguous

In Continguous
we load the entire process in one part of memory.
(in continguous memory location)

Non-contingous
we breack the process into chunks and then store it into memory, where available

Contingous

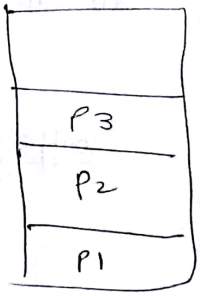
Static
(first decide degree of multiprogramming, then divide Memory into degree parts
M/N,
and store process,
(Advantage:-
Simplest way of implementing)

Disadvantage
↳ Space will be waste
↳ If one process is longer, then it will not be executed.

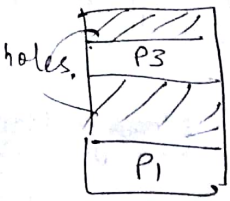
Dynamic
we will not predivide the memory into parts.
When, a process come, we will store in memory

(Degree of multiprogramming
(how many process will store in RAM at particular time)

Ex

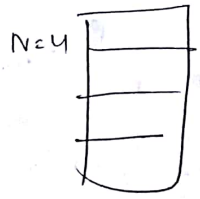


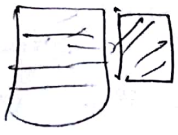
P2 terminates



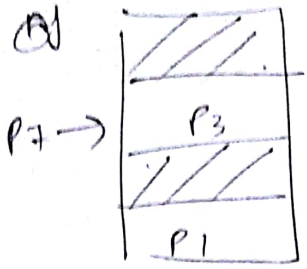
After P3, executes,

duty to merge Consecutive holes





Initially, the whole RAM is a hole (single hole)
It is the job of OS, to maintain the information of where all Process have (in memory).



Where to store.?

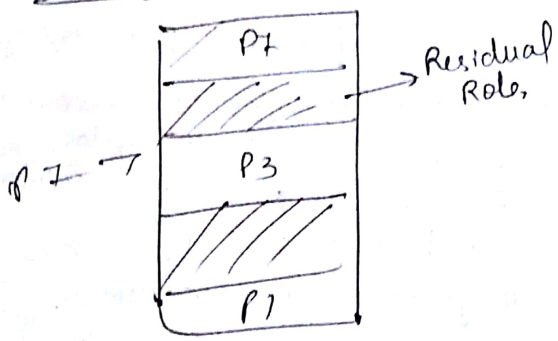
Strategies

(1) First Fit
Where the process first fit, we will store in it

(2) Best Fit
We will store which is larger than P7, but less than all larger.

(3) Worst Fit.
We load the process in the largest hole.

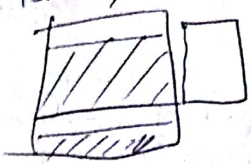
Advantage →



(Residual hole will be larger than Best and First Fit

Disadvantage of Contiguous
External Fragmentation

wastage of small, small memory

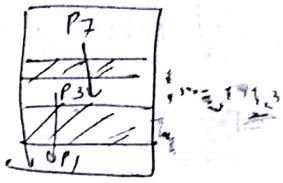


even though we have space, but we cannot save it.

Solution?

& we can shift ~~holes~~, Process at one part of memory

Compaction
↳ expensive process.



Numericals can come
(Easy)

In next class → non-contingous
Paging

19/3/2018
Class-27

Segmentation 8086 ↗
Flat address space
↳ whole space is similar:
Can use any memory instruction on any address.
(can be used for both data & memory address)

LDA 8000 H

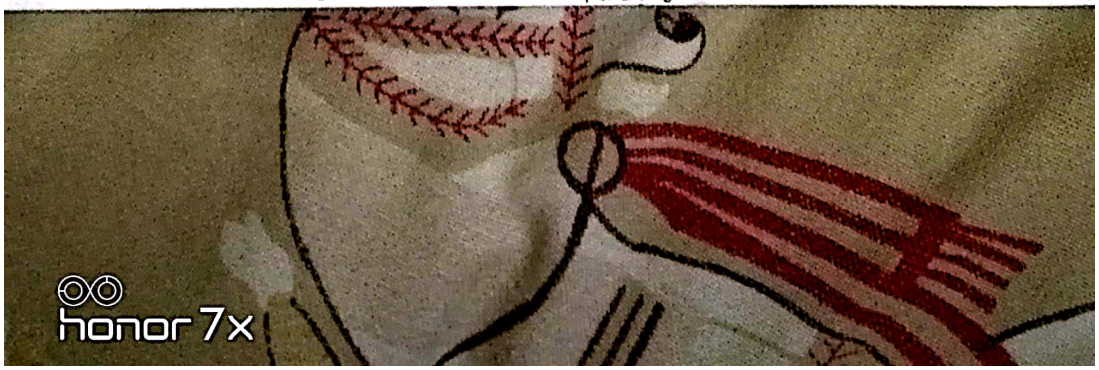
It is a address space.
00000 H → FFFF H

Stephens Paul Morse.
'felt that 8085 is good for Assembly language.
but not for high level language.' (No difference b/w part of memory loading data or part of memory holding code)

relative addressing mode
↳ content of we add the PC to the address given in the instruction.

~~Offset~~ ~~relative~~ (Jump & Call Instruction) → use relative addressing.
Relative Address - 2 positions ahead / back
(Jump)

In portion of memory, we might want to use relative addressing,
In portion of memory holding data, we might need to access Array or ~~string~~ string.
(Different parts of memory access differently depending on whether it holds data or code



→ How can you implement that in microprocessor?

Morse proposed Segmentation → useful when we need write program in HLL.

↳ developed Intel 8086 (First MP to use segmentation.)

* Registers in microprocessor today core a superset of register used in 8086.
* Much powerful microprocessor.
*
* Several others (8088 ..) were advanced version of 8086.
* Instruction set of others is a superset of 8086.

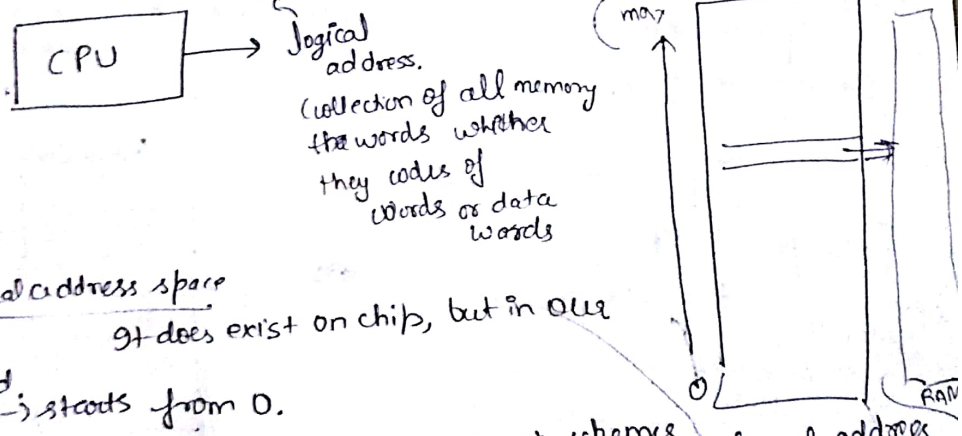
The whole family of Microprocessors influence by 8086 is called x86 architecture

makes use of segmentation and instruction set is called x86 instruction set. (Basic philosophy of programming same as 8086)

Now,

What is segmentation?

Segmentation is a memory management scheme where the linear memory is logically partitioned into segments each having unique purpose.

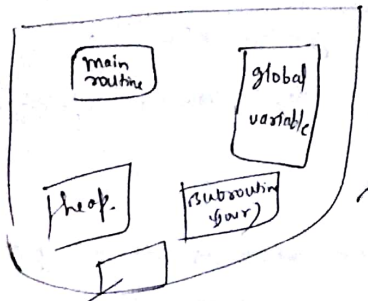


Logical address space

it does exist on chip, but in our mind → it starts from 0.

There are many memory management schemes acc^n to which we can allocate the logical address space on physical address space

→ Segmentation has nothing to do with mapping.
→ Segmentation partitions the logical address space into some segments and each segment contains some unique part of process.



Logical partitions
size of partitions can be varied widely.

we can have stack segment.

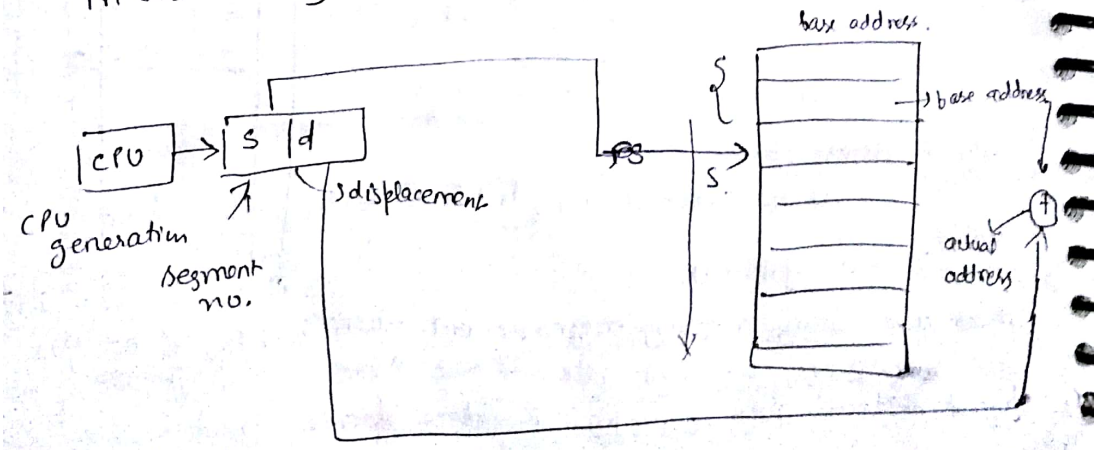
Advantage → Optimise the memory access easily.

Ex- main (normally) contain code
(To access code easily, access main part)
⇒ can have more than one code segment/data/stack segment.

Whether we do segmentation or not, we need to do mapping, whether contiguous or non-contiguous memory allocation.

Segment Table

If we segmentation, then there will be segment table in the memory.



Segment no., used as index,

displacement → (what is offset of word, within that segment)
(where is word stored in that segment)

OIVS ; find base address
adder (base address + d)
= actual address.

→ Processor should have necessary hardware, support to implement segmentation.

→ What are the commonly used segments?

(1) Code (CS)
It will store the code (Machine language program)

(2) Data (DS)
will contains the global variables of the program.
(Heap can be accessed in both DS & ES) (for Dynamic allocation)

(3) Stack (SS)
program stack will be there, local variables

(4) Extra (ES)
(similar to DS) (Extra data segment)

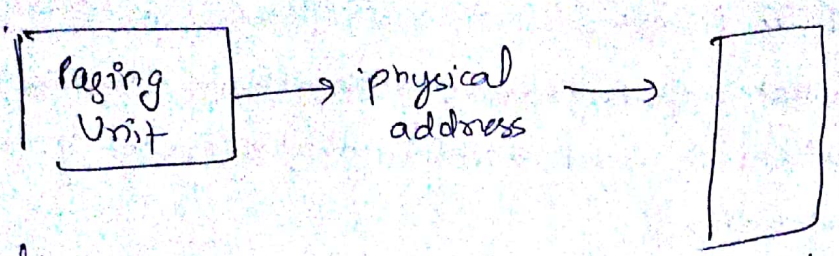
Ex- copy one array to another place, we can access these 2 segments simultaneously. We can access memory at diff. places simultaneous (DS & ES)

In modern day's processor, they use combination of segmentation & paging, (though both can be implemented independently)

CPU generate logical addresses

CPU → logical address → Segmentation Unit → Linear address →

I S/d



linear addr: obtained after segmentation starts from 0000 to max

Physical addr: Actual address space where memory word is stored in RAM.

Intel Pentium (Microprocessor)

Not very simple and not very complex.

Before this,

8004, 4004, ... 8085, 8086 (has numerical name)

then Why Pentium? (as invention in USA only if it has a name, not a number)

for legal issues.

→ Microprocessor can have upto 16k segments free process (but not necessary to fill them)

but microprocessor can not access all the segments at a particular point of time

→ only 6 segments can be accessed

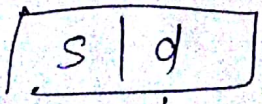
↳ Because there are only 6 segment registers

Code, Data, Stack and 3 extra

→ Microprocessor doesn't know, which segment has which type? Need to specify.

→ segment can be of upto 4 GB.

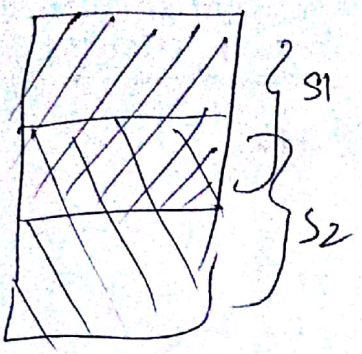
Base address of 6 segments can be stored only in these registers



d should be b/w 0 to upto 4GB size

→ If we want to access any outer segment. other than 6, load the segment base address into register, Each register will have different segments.

Segments can be overlapping



→ Intel Pentium also use paging

→ If we are installing OS on this MP, then ensure page size.

→ Intel Pentium uses segmentation & paging.

→ page size will be fixed
  └ 4KB and 4MB.

Not much questions

Generally Question can be short note on segmentation

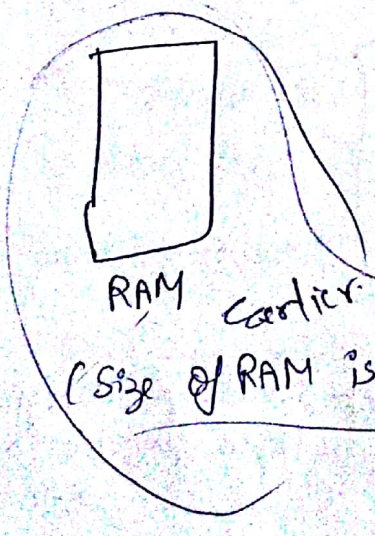
(zada nahi aata)

Three things that do exist but are not visible ⇒

1. Family of Ghosts.
2. True Love
3. Virtual Memory

* Virtual Memory

→ Earlier day, no virtual memory.
→ Load process in RAM & execute.
→ Size of RAM is fixed. To increase degree of multiprogramming, it is fixed.
→ If size of process > size of RAM, can't execute that process.
∴ we use virtual memory.



Earlier → (Size of RAM is fixed)

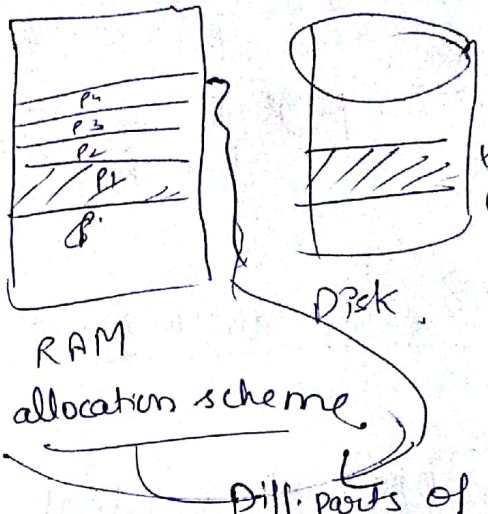
→ We donot say programmer that 8 GB is your limit.
Give freedom to programmer do write very large addresses.

The address space that programmer assume RAM at time of writing the program. The illusion that programmer has while writing the program is virtual memory

~~If the degree of multiprogramming is 1 then~~ ~~find out~~.

Advantages

→ can have multiple processes in RAM.
→ Increase degree of multiprogramming.
→ Give freedom to programmer to write long program.



For virtual memory implementation, we use non-contiguous memory allocation scheme.

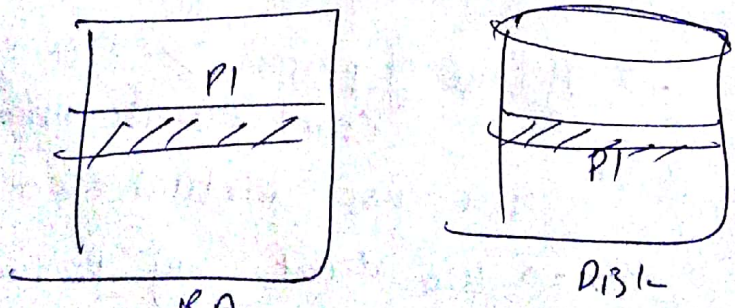
Diff. parts of processes at diff. portions of memory.

→ In virtual memory, do not keep all portions of program in memory.

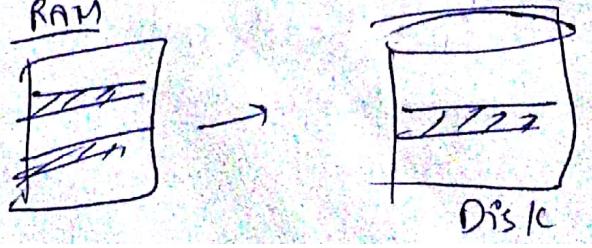
Only those portions of process are kept in RAM, that are currently been executed. The remaining part of process is stored in disk.

(Size of disk is very large. Since we use non-contiguous memory location, this is possible.

This is the concept of virtual memory.



For multiprocesses running,



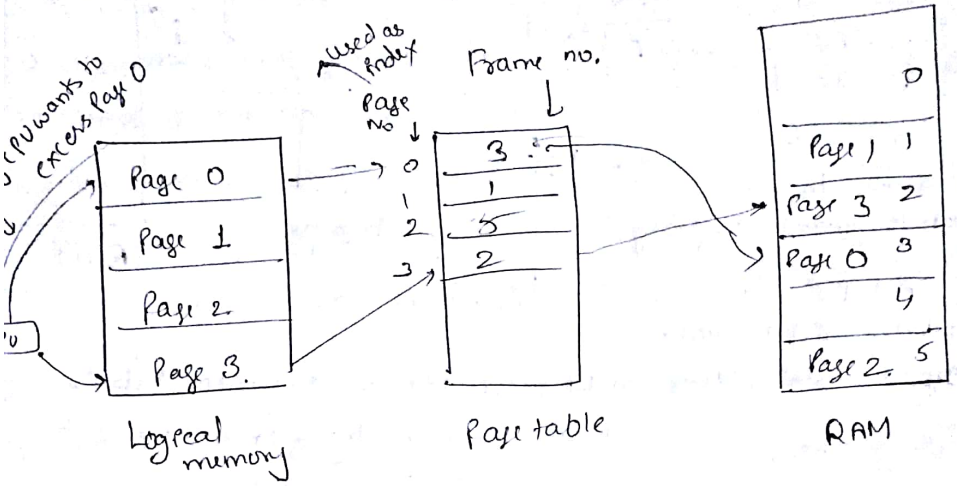
keep small parts of multiple processes in RAM.

(Not easy to implement virtual memory)

20/3/18
Class-28, 29

Paging

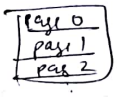
↳ Non-contiguous memory allocation scheme



logical memory → address space of process

→ whole footprint of process in memory.

we divide this address space in different equal sized units called pages.



Pages are logical units (not in real)

(we have to store them into physical memory RAM)

RAM is divided into equal size frames,

↳ size of pages and frames are same, respectively.

↳ Any pages can be loaded at any frame in RAM due to equality and size.

Data structure: Page Table → to keep track of which page link to which frame

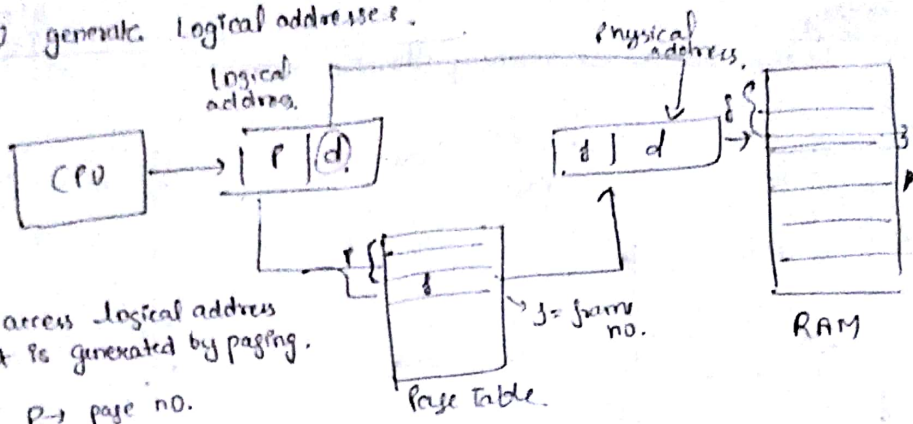
Page contains code & data.
(contains all addresses)

Page → Logical entity → (doesn't exist)

Frame → Real space in RAM

Addressing is always relative in paging

CPU generate Logical addresses.



CPU access logical address that is generated by paging.

p → page no.
d → displacement

Convert logical address to physical address using page table.

Page size is fixed,
There is no external fragmentation.

Now, we are trying to access 5th frame in RAM.

Size of page = Size of frame

we can access d^{th} byte in that particular frame.

Once install an OS, page size becomes fixed.

In most OS,
page size = 512 B to 16MB
acc" to (8th edition)

Advantages of Paging

⇒ No external fragmentation
because size of pages and size of frames are exactly same
(No wastage)

⇒ Size of process can be independent of page size.
(Multipages can be used for large process)

Let us assume page size = 1 MB.
process size = 10 MB. 10.5 MB

10 pages required

11 pages required

0.5 MB is wasted.

It is less lossy and there are ways of handling it)

Is known as (internal fragm

For every process, there is a page table in system
↳ stored in RAM.

↳ As many page tables, as the no. of process in multiprogramming

→ How to keep track of page tables?

We have a register called pagetable base register (PTBR)

→ holds the starting address of the page table of the process which is currently running.

→ when context switching, we have to just change the value of (PTBR).
↳ ∴ makes context switching very fast.

→ (Q) If we want to use LDA (basically Loading) instruction, how many times we have to access the memory?

2 times.

(1) First to access the page table,

(2) Access physical address.

↳ and then find the location and ~~load in that~~

↳ load in that particular location

→ it takes twice the time than continuous memory location

LDA 8000 H → Only 1 memory access

LOD in paging: we need absolute address and hence access RAM page table to get the frame number which is absolute address and then use frame no. for actual reading/writing.

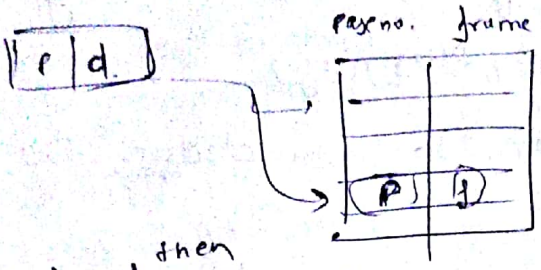
RAM is much slower than processor.
(2 memory access needed)

You are taking more time in ~~read and write~~

Paging has many advantages hence we must use paging.

→ Use one special hardware chip:
(Transition Look-aside Buffer (TLB)

↳ dedicated & fast hardware chip. (to improve performance)
↳ associative memory (content addressable memory)



In this, pages nos of processes that are accessed frequently and their frame no. are stored.

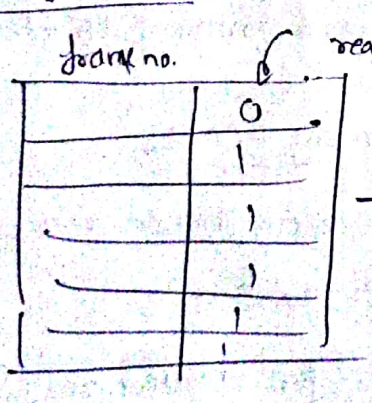
If p is found, then TLB hit else,
↳ TLB miss,
↓
then go to page table.

→ If we get P here only, we read, without accessing page table.
→ It is very fast
→ It is small in size.
(not all entries can be stored, only recently accessed)

Effective Access Time = hit ratio × [TLB access time + memory access time]
+ (1 − hit ratio) × (TLB access time + 2^x memory access time),

→ If hit ratio is high, performance is very good.
→ Access time is very less than memory access time.
→ If hit ratio is ——, then small effective access time.

Page Table

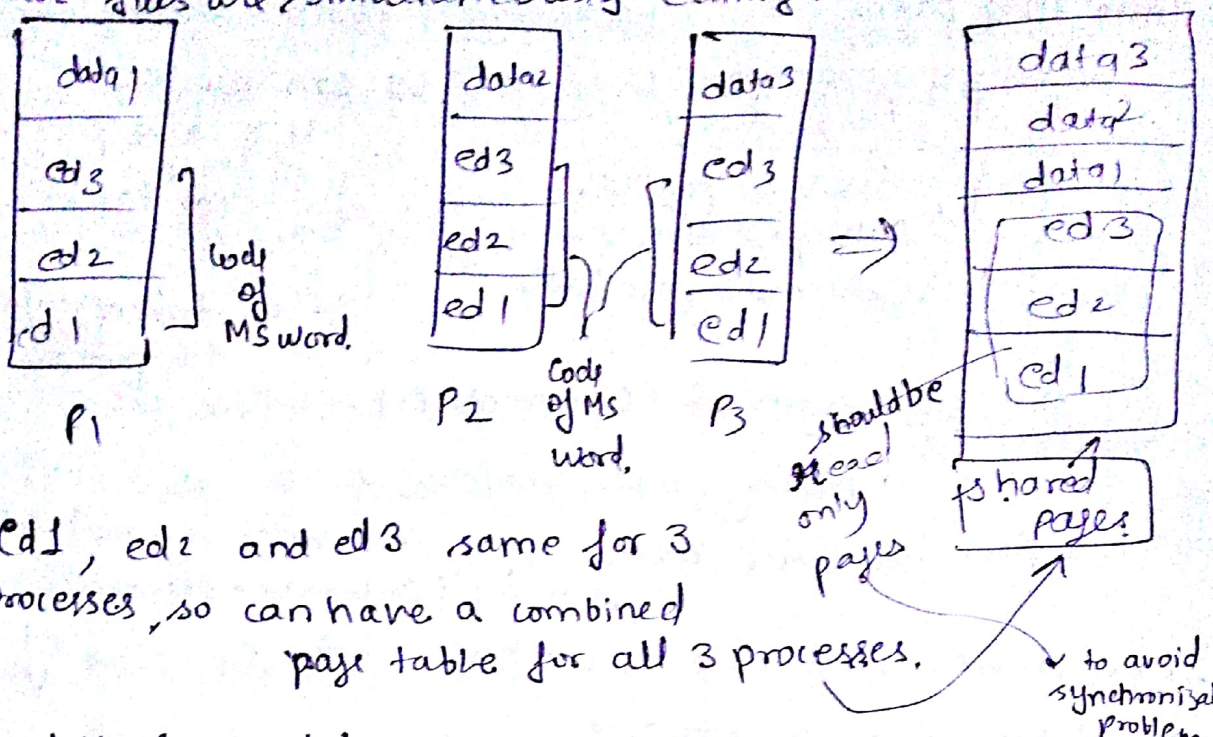


→ We can augment page table with a new bit called read only bit.
→ Initially rob are sets.

→ When processor make any change in a page, then rob = 0.
→ we have to store back to disk, in case page is modified.
↳ This is to ensure, disk has modified data

→ In case no modification
rob = 1.

& 3 doc files are simultaneously editing.



ed1, ed2 and ed3 same for 3 processes, so can have a combined page table for all 3 processes.

⇒ Page table is stored in some page of OS.

→ This might take many pages to store page table,
→ To access page table, we need difficult methods.

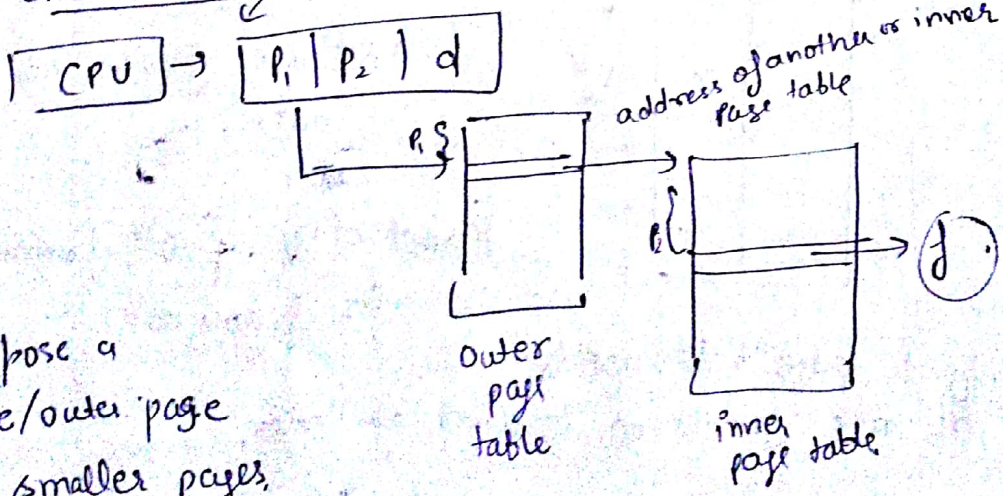
There are two techniques → to ensure pg table is not very large,

1) Hierarchial
2) Hashed page table (Not much important, read from book)

Hierarchial Page table

Two-level paging

CPU will generate logical address of this form.



Decompose a large/outer page into smaller pages.

one

can be multiple page table.

Three level paging

P_1	P_2	P_3	d

↳ inner page table
↳ outer page table
↳ second outer page table

we can also use 4, 5-level paging for even larger code.

It is talked only for one process.

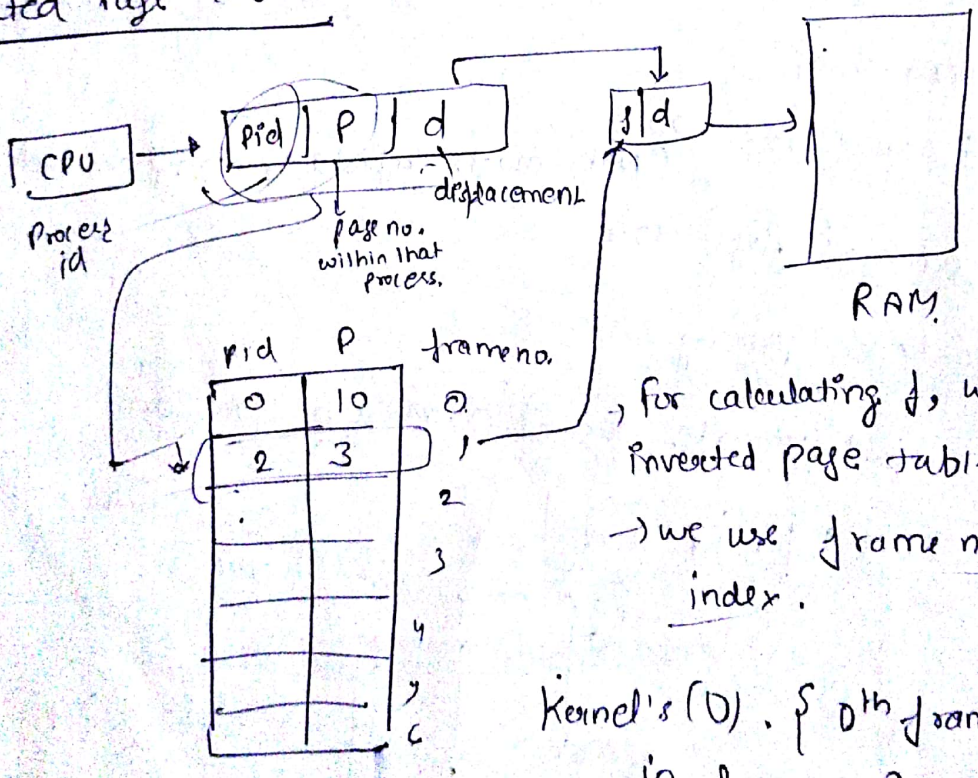
Problem with hierarchial (Disadvantages)

↳ Memory access increases
Since you will have to access memory multiple ↳ (Performance degrades) times.

→ programmes will have to comprise between size of page table and performance.

Every process has its own page table and stored in RAM and hence an alternative for this.

Inverted Page Table



we use frame no. as index

→ For calculating f, we need Inverted page table.

→ we use frame no. as index.

Kernel's (0). 5 0th frame stored in frame no. 0.

Advantage: only 1 page table used.

→ Inverted because frame no. is used as index.

Problems:

↳ No shared pages (as process id of every process is diff.)

↳ You can not implement virtual memory. (Also, take linear time as compared to O(1))

UNIT-6 VIRTUAL MEMORY MANAGEMENT.

1, allows execution of process that are not in RAM.

↳ *process partial in RAM.
* can be executed using virtual memory.

Advantages

↳ any arbitrarily large process can be executed. (size of process not limited by size of RAM)

↳ degree of multiprogramming increases.

→ How you implement virtual memory?

There is a technique called,

Demand paging

1, most widely used technique to implement V.M.

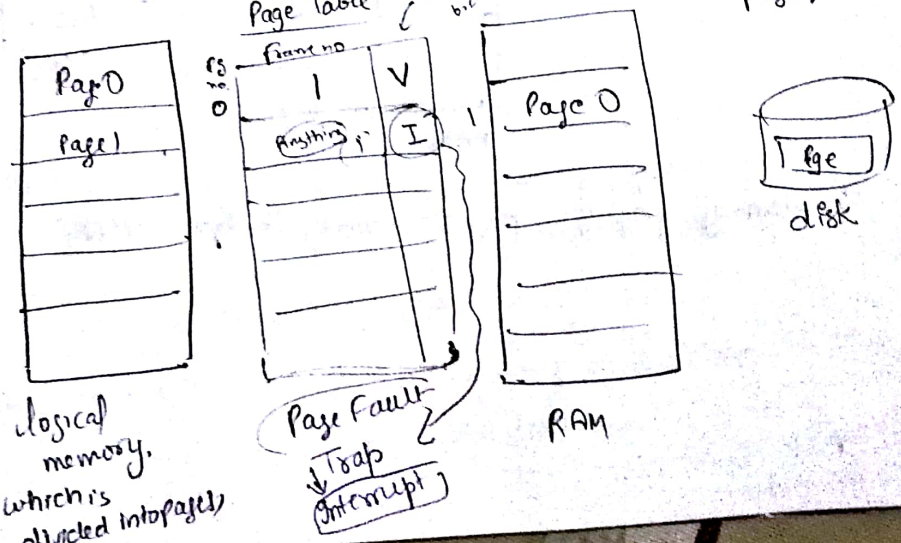
↳ Based on paging.

DEMAND → we load into the RAM, a page only when it is demanded by processors (CPU).

Some pages may never be loaded into RAM.

pager = Lazy swapper

(load program from disk → RAM) → only when CPU demands those particular page.



→ Some page in RAM & some is in disk.

How to know process has many pages and which pg is where?

→ By changing Page Table.
Valid/Invalid Bit.

→ If CPU wants to access a pg in RAM, then execution continues as it is.

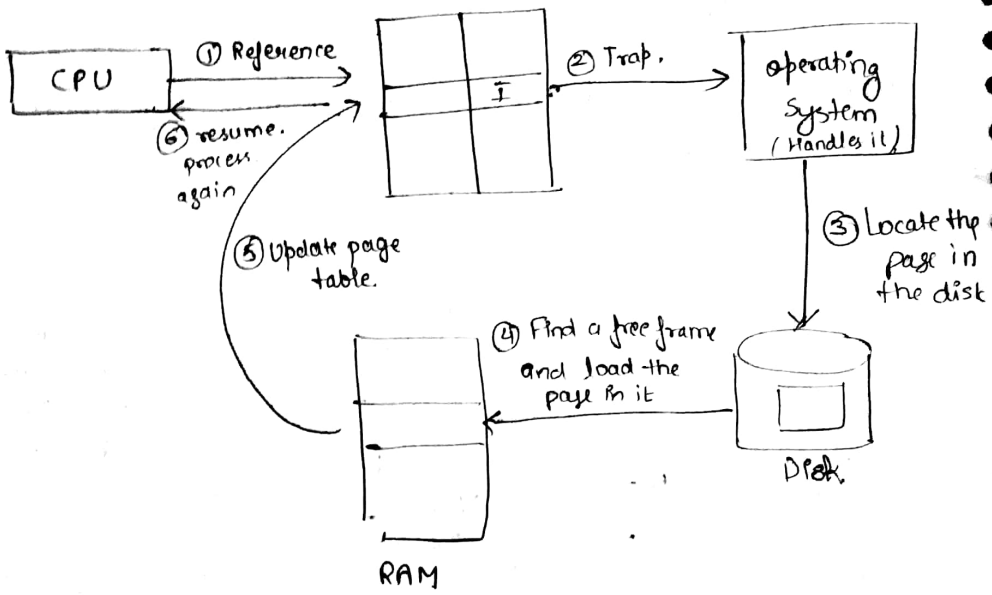
→ If CPU wants to access a page, which is not in RAM, (if finds that valid = I), there is an interrupt

system generates → page fault trap

Trap → highest priority, non-maskable hardware interrupt

4. How to handle interrupt? (5-6 marks Question)

Handling Page Fault —



On demand paging, pages already loaded in RAM

Pure Demand paging. (extreme/special case of demand paging)

↳ we start a process with none of its pages loaded into RAM, and we start executing. (load contents from PCB and start)

Just in the starting, there will be a page faults

Advantages

→ OS does not assume which will be used. The pages which will be actually be used will be loaded,

→ Response time quiet low.

(CPU is executing fetching instructions from one locality of mem.

Locality of References. (also seen in case of second memory & cache memory)

↳ for code (instructions)

↳ for data. (we will be doing load/store in one locality of memory)

Most computer programs exhibit this phenomena.

→ Global variables (stored in one part/locality of memory as variables stored in one part.

Here we talk worst paging.

→ If we want to implement virtual memory, disk, paging related h/w needed.

Let us define,

Page fault ratio is p. = $\frac{\text{No. of page faults}}{\text{No of memory access}}$

$0 \le p \le 1$, $p \to 0$

Effective access time = $(1-p) \times$ memory access time

$+ p \times$ page fault service time (more time as it is disk access)

P ↓ then,

effective access time = memory access time.

we have to minimize P, tried

we need to keep page fault service time small

4th step in 6 step process

If we did not find a frame, then we have to use

* Page replacement.

Copy one page to RAM and remove the page from frame to disk

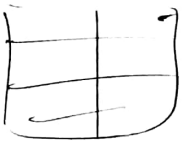
(When no free free frame in RAM) (which should happen)

↳ steps :-

→ Back the old page in disk.

→ Put new page

→ update page table for both incoming and outgoing page.



read only bit / dirty bit / modify bit / alternate name

If the page has not be modified yet, then no need to store back to disk, as old copy is available as it is.
→ save time.

For Page Replacement, we use two types of algorithms.

(OS uses 1 of many types of algos)

Page Replacement Algorithms ⟶ parameters → Page fault ratio,

(Try to minimise this)

To measure page fault ratio, we use reference string.

sequence of random numbers

1, 0, 2, 4, 0, 1, 4, 5, 7

page number

other type → Frame allocation algorithms

When process will be given how many frames?

(triggered as long term scheduler)

→ we want to decide which page to remove from RAM & store in disk.

→ The victim is chosen by page replacement algo,

→ Page replacement algo used very frequently even frequently then CPU scheduling also.

→ had to be very fast.

→ Assume that a process wants to access page no-1, page no.0, 2..

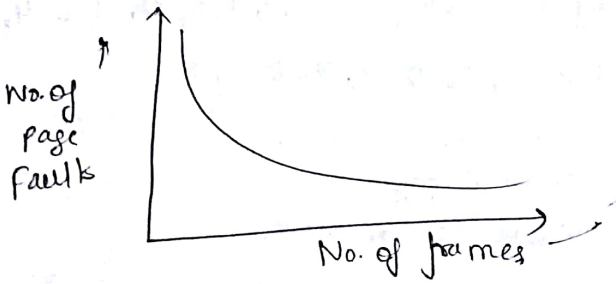
→ different random reference string run on replacement algos,
and then avg. values are taken.

→ No. of page faults suffered by process depends on page replacement algo

→ It is common for all processes

→ There is another factor on which page fault depends:-
no. of frames.

(If we allocate less frame, then large Pg. faults,
if allocate more frame, than small pg. faults)



If we allocate too many frames to a single process, hence degree of multiprogramming decrease.

→ If we try to ↑ multiprogramming, by allocating less no of frames to each process, then large no. of page faults will occur.

If no. of frames allocated > no. of pages
then, page fault = 0.

28/3/2018
Class-30

Two types of algorithms
→ Page Replacement
→ Frame Allocation.

Page Replacement algorithms.

① FIFO page replacement algorithm. —

First In First Out

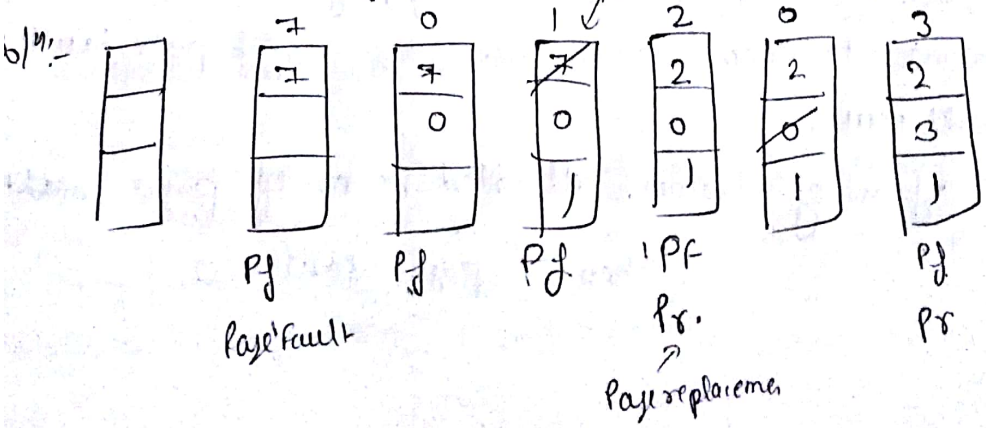
Whichever page is loaded into memory first, will be replace first.

→ do not need to load the time when loaded.
→ just maintain a queue.
→ It is the simplest placement algorithm.
→ Performance is not good.
↳ page fault ratio/rate increases.

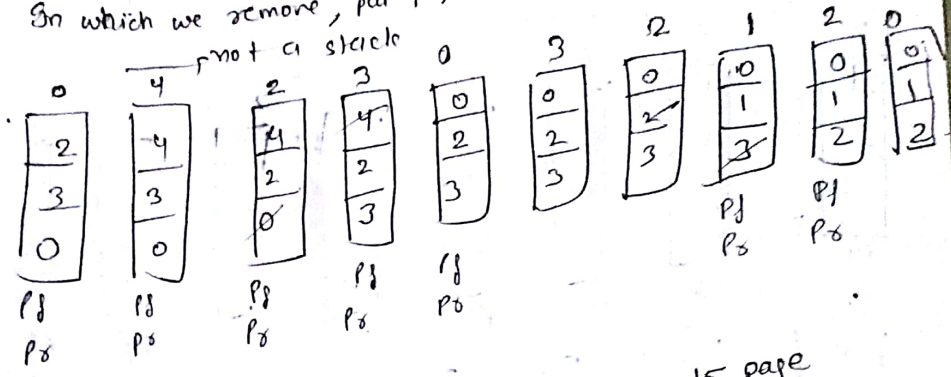
Eg:- A process is allocated 3 frames in the memory and has given a reference string = 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.
↳ page numbers.
There are atleast 8 pages,
↳ there can be more.

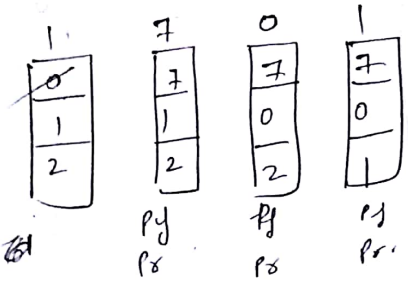
→ Calculate the no. of page faults. (We use demand paging)

b/w:-



In which we remove, put page in the same frame, not a stack



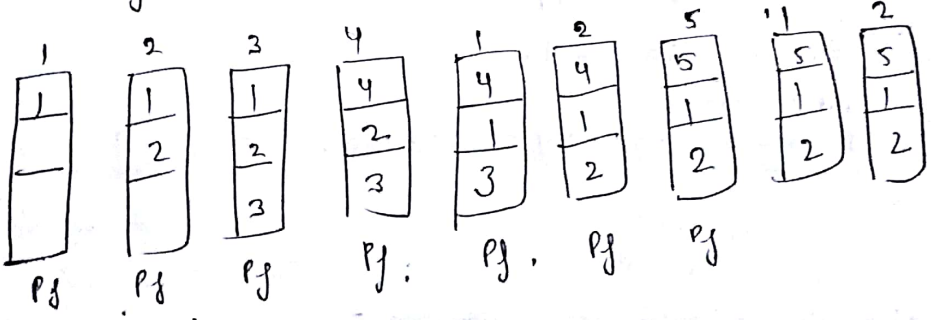


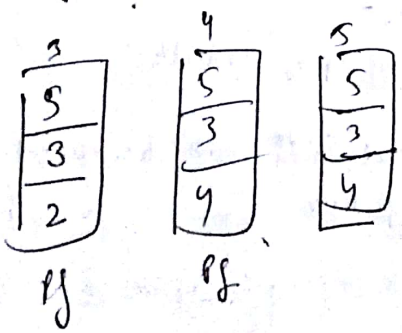
You have 15 page faults.

2nd Numerical

Reference string:- 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

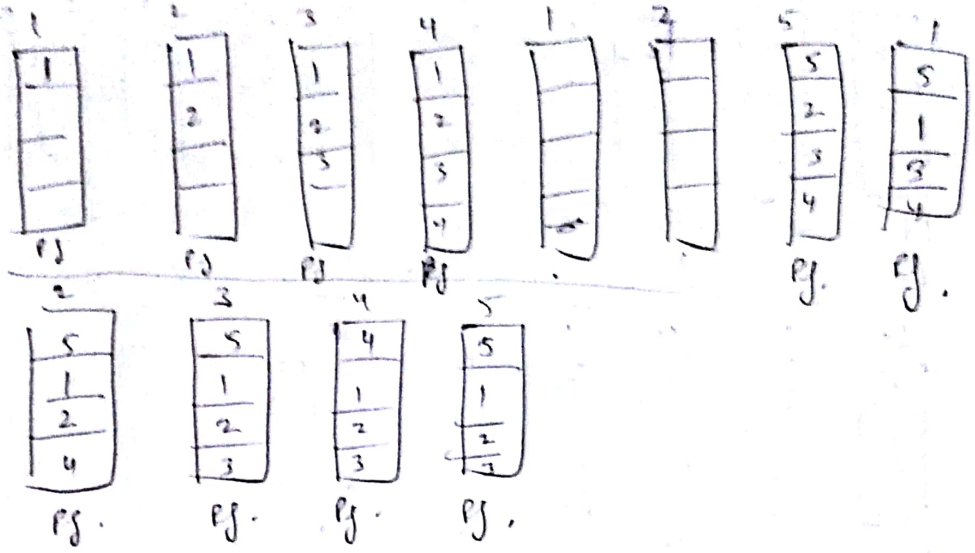
No. of frames = 3. Calculate no. of page fault





9 page faults.

when no of frames = 4



10 page faults

No of ~~frames~~ page ~~faults~~ fault should be decreased, when we increase frames, but it is happening opposite here,

∴ This is the problem.

FIFO shows,

Belady's anomaly — parado>

-ve thing
drawback.

(3) Optimal Page Replacement algorithm. —

It gives the minimal no. of page faults. because

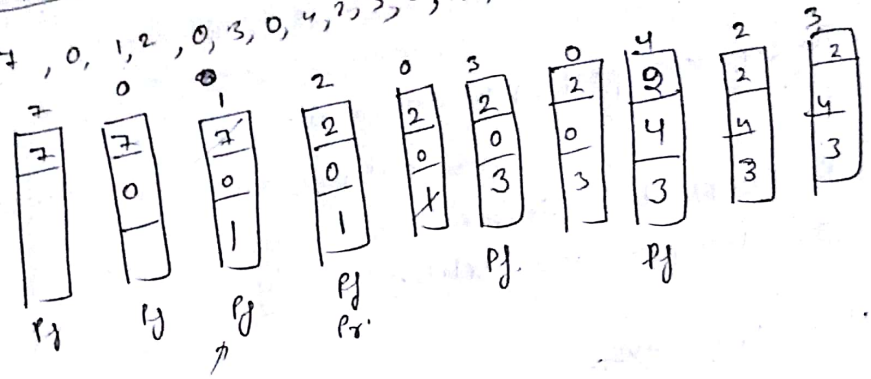
Here, we replace the page that will not be used for the longest period of time, in the future,

and there is no,

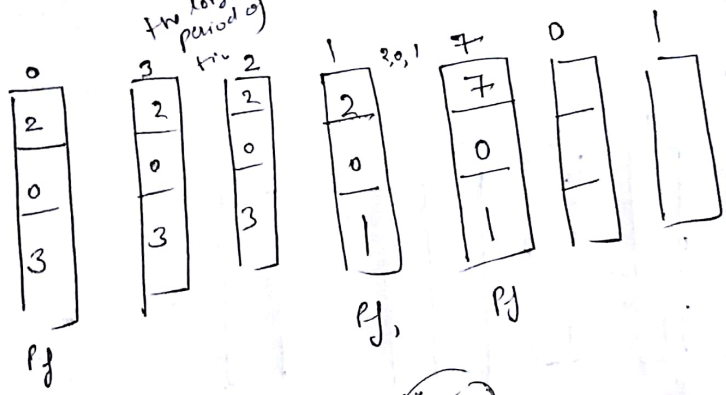
Belady's anomaly. here

Solve, same numerical,

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1,



7 is not being used for the longest period of time



No. of page faults = 9 (15) FIFO.

Main Problem

↳ Not implementable

↳ How would you know that which page will be needed when

↳ theoretical only.

↳ This algo is taken as a datum (piece of information) for comparison, to compare performance.

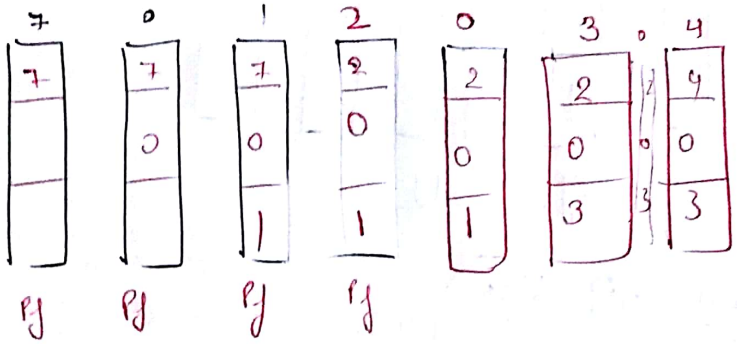
(3) Least - recently used (LRU) page replacement algorithm –

99 % chance to come numerical in this →

In the near past, page did not use
→ remove that page

- Possible to implement
- Does not show Belady's anamoly.
- two ways,
  - Use counters. using
  - using stacks

Solve Numerical



2 page faults

If a page is access, quiet frequently, possibility is that page will never be replaced, like 0 in this example.

① Counter-based implementation —

↳ Used as a logical clock,

Page table will have extra column, 'time-of-use' field.
integer type
not → datetime ✗

whenever, a CPU is accessing any page, counter is increment.

→ When a reference is made to the page, its time-of-use is set to counter. for a particular page, time-of-use = counter,

→ The page with the lowest value of (time-of-use) field will be replaced

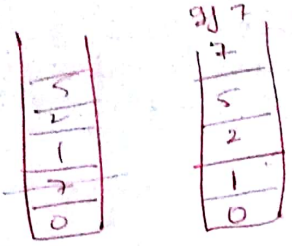
→ you have to make sure that this counter is not overflowing.

② Stack based implementation.

We use data structure that looks like stack (but not actually stack)

→ we maintain a stack of page no's that are being referenced,

→ When a page is referenced, we remove it from stack and place it on the top of the stack



If 7 reference → LRU page will be at the bottom

Two more algorithms left

2/4/2018

INPUT-OUTPUT MANAGEMENT. (Question as short note)

Computer will be performing processing & I/O.

there are some processes which are called I/O bound process. (maximum time - I/O)

If there are such processes, then I/O management become more important.

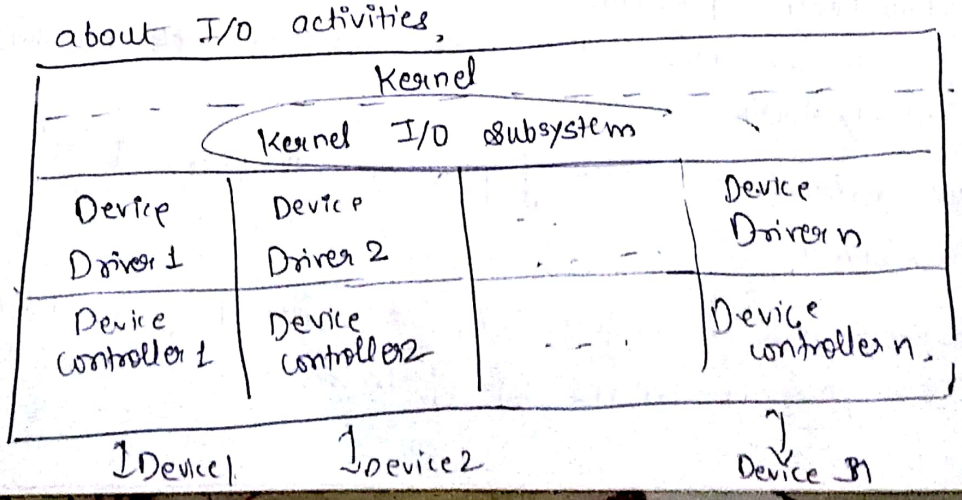
Kernel of OS has an I/O subsection

→ is part of a kernel which interacts with the input output devices.

→ responsible for all I/O management.

advantages

→ The rest of the kernel did not need to worry about I/O activities,



I/O subsystem uses device drivers.

Device drivers are programs.

→ They act as an interface b/w particular device and OS.

→ They are software.

→ Device drivers are device specific.

→ printer has different.

→ camera has different.

→ job of device driver is to provide uniform interface to the I/O subsystem.

Device Controller

are implemented in hardware.

→ hardware part through which devices are connected.

→ to connect I/O device to your computer.

I/O devices are of different types.

Types -

→ (1) Storage devices
(they are part of memory hierarchy)
External hard disk, USB drive.

→ (2) transmission devices
Ex - Modem
which allow your computer to send or receive data.

most common form of I/O devices.

③ Human-interface devices.

all users can provide input to the computer and output can provided to the user.

Input device → Keyboard. } Most common.
Output device → Monitor }

Printer, touchpad, Joystick, (touch screen.) both input & output.

speakers, earphones, scanners, cameras.

→ how smartphone screen works?

resistive type

capacitive type (In our smartphones) (capacitors).

④ Specialized device

they are not used by general users.

For specialist.

Ex- asteranauts,
biologists.
MRI machines, CT scans.
flight control computer of aircrafts.

Hardware

what are different hardwares for I/O management

① Port

Most common — USB port.
Earlier - different type of port for different, for Printer, ...

② Bus.
collection of wires.
Through these buses, data and control information are flow.

③ Controller
→ electronic logic unit
controls the bus and the port.
Ex – MUX, tri-state buffer,

I/O ports contains four register.

① Status register
↳ tells the current status of the device.
(how many pages no, left pages)
↳ read only register.

② Control register
↳ processor write the command or control word.
Ex: - how many pages to print, speed of printer etc,

③ Data in register
When computer is taking input, data in register acts as buffer and is stored here.
Processor reads from here.

④ Data-out register,
Processor places data bit in the register.

→ You may require subset of these registers.
not all used.
For example → for keyboard
data-out register is not used.

On many devices, status register and control register can be multiplexed. Similarly, data in and data out can be multiplexed.

Modes of I/O –

① polling.
lowest possible technique.
→ processor keep checking input register or other for data in loop to check available data.
↔ called (busy-waiting)

② Interrupts
when any input is available, then interrupts occurs, otherwise processor keeps busy in other work.

INTEL 8085 has 40 pins.
5 pins,
RST → 5.5
6.5
7.5
TRAP.
INTR,
1 acknowledge INTERRUPT

SOFTWARE INTERRUPTS
8
5.0 to 5.7

③ DMA
(Direct Memory Access.)
who is accessing the memory directly?

copying can be directly from input devices to RAM or RAM to ⊕ input devices,
(in other, first place in register (intermediate))

DMA
↙ burst mode.
↘ cycle stealing

Read More (in)

Complementary devices :-

(1) teletype = keyboard + display.

In some computer, there is one device driver for one teletype.

(2) photocopier = Scanner + printer.

Virtual devices —

→ PDF printers. (Ex - in MS Word)

Properties of I/O devices —

→ character stream vs block (or chunk)

character by character (dot matrix printer)

blocks of data (hard disk)

→ sequential vs random access

(one by one in a particular order) (Punch cards, tapes drives)

(any of the byte can be accessed) (Most of the pendrives)

→ synchronous vs asynchronous

One byte send, if send then other byte. (If devices are fast, then use synchronous)

(all send at a time) (For slower devices)

→ shareable vs dedicated.

(monitor) If (UI based OS, then dedicated (difficult to share

Printer is non-shareable

→ speed.

Vary widely ? processing amount of data varies.

Processing speed varies,

(e.g.) Scanner do very fast.

If we ~~write~~ type, then take time.

→ read-only, write-only, read-write.

Some only input devices,

some only output devices, others act as both.

Services provided by I/O subsystem —

① I/O scheduling.

If two pages are downloading then which page download first?

② Buffering

creation, maintainence of buffer is a job of I/O subsystem.

Buffer is temporary storage.

③ Spooling

MS Word → Print

MS Excel → Print

Special type of buffer → spool.

~~Printer is a dedicated device,~~

If data is sent to printer in interleaved fashion, ~~then the~~ then when one process has completed sending data, it will print.

Spooling is responsibility of I/O subsystem to manage dedicated devices.

spooling more important than I/O scheduling.

④ Error handling

If you are trying to print, and there are no pages in the printer, then message occur,

So, it is responsibility of I/O subsystem.

Error Handling.

⑤ I/O protection

Processor should not misuse I/O devices.

I/O subsystem handle this.

(You read all these stories in the book) (No numerical)

(short question can come)

5/4/2018

Stack algorithm:

We have studied three algorithms before.

→ They satisfy some sort of stack property.

→ They do not suffer from Belladys anamoly.

for all condition, always, all position time

if you can show this property,

pages in memory for n frames $\subseteq$ pages in memory for (n+1) frames.

then that page replacement algorithm is stack algorithm.

Test this for LRU algorithm

n frames: n most recently accessed pages = S_1

(n+1) frames ;· n+1 " " " " " = S_2

$S_1 \subseteq S_2$ (Clearly)

Hence LRU is stack algo

LRU does not suffer from Belladys anamoly.

Also can be shown for Optimal page replacement.

(4) LRU Approximation Page Replacement Algorithms —
(not exactly LRU)
principle is LRU.

There is one column of reference bit in page number

when a new page loaded reference bit is cleared to 0

Reference bit = 1, when page is accessed (either for read/write)

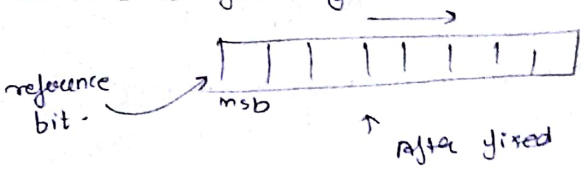
Reference bit = 0, when PG is not accessed

(a) Additional Reference bit page replacement algorithm.

→ We use a 8-bit shift register for each page table entry.

After fixed interval, all register are shifted to right.

In MSB we fill reference bit



↑ After fixed interval of time

LRU page = min. value of the shift register

There can be a tie,

To break the tie, we use FIFO.

(b) Second Chance page replacement algorithm

→ It works like FIFO.

→ If ref. bit == 0, then replace it.

however,

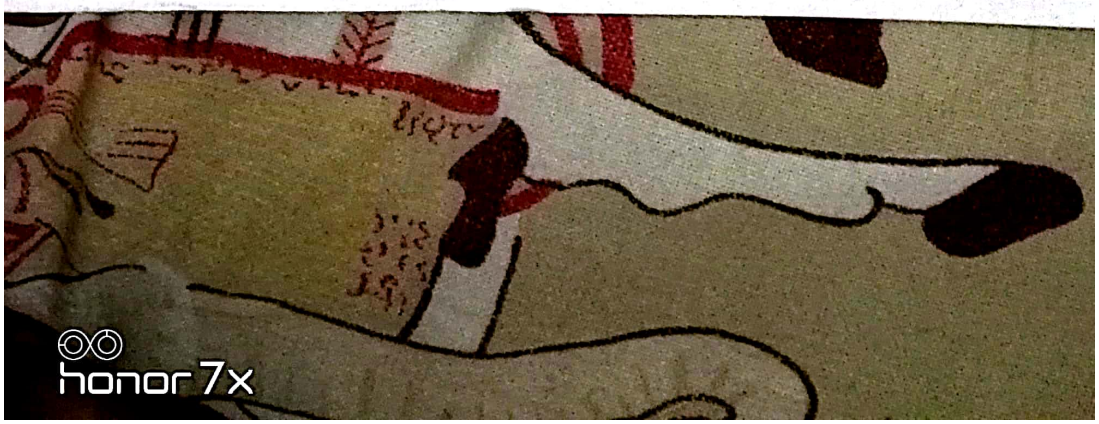
ref bit == 1, then give it a second chance
↳ do not replace it
↳ ref bit ← 0

Good thing about it,

→ If a page is frequently accessed, then that page will not be replaced

→ It is feasible or quite easy to implement

Queue is maintained for all pages, like FIFO (not exactly FIFO)



(c) Enhanced Second Chance PRA.

(Better performance)

If a page has not modified, then no need to copy page into disk.

Pages which are written over are bad candidate for replacement
↳ concept.

Again it works like FIFO.

We use reference bit, modify bit

∴ we basically have 4 classes of pages,

class → 00, 01, 10, 11

00 → not referenced but work
01 → not referenced but only read access
10 → referenced but work
11 → referenced but only read access

We check.
from
front → rear,

→ try to find out class of 00,
first page we find with class 00, we replace it,

→ if did not find,
then search for 01

→ if did not find
then search for 10

→ then 11

This concept known as enhanced second Chance.

5.) Counting Based page replacement algorithms –

→ Counter for each page

→ Counter is initially cleared to 0

→ Every time page is reference, counter is increased,

If page fault occur, then which page replace,

5a.) Least Frequently Used (LFU) page replacement algorithm.

Not LRU

Replace page with minimum value of counter.
as it is referenced the least.

5b.) Most frequently Used (MFU) page replacement algorithm.

Replace page with maximum value of counter,

→ Concept behind this, is that, it thinks that which page has used most, will not be needed later

→ Page having least counter, may be just arrived & and needed more later,

Performance of both of them are bad.

Both are expensive as use counter in both of them

5.) Counting Based page replacement algorithms =

→ Counter for each page
→ Counter is initially cleared to 0
→ Every time page is reference, counter is increased,

If page fault occur, then which page replace,

5a.) Least Frequently Used (LFU) page replacement algorithm =

Replace page with minimum value of counter.
as it is referenced the least.

X Not LRU

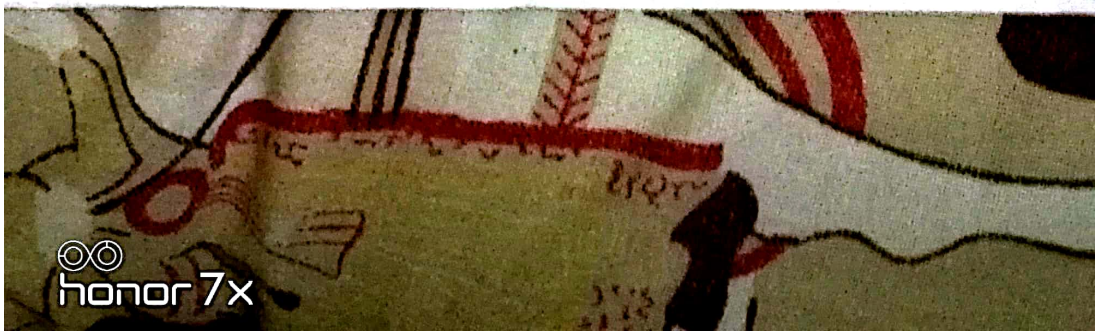
5b.) Most frequently Used (MFU) page replacement algorithm.

Replace page with maximum value of counter,

→ Concept behind this, is that, it thinks that which page has used most, will not be needed later
→ Page having least counter, may be just arrived & and needed more later,

Performance of both of them are bad.

Both are expensive as we counter in both of them,



Numerical

Reference String,

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Frames = 3,

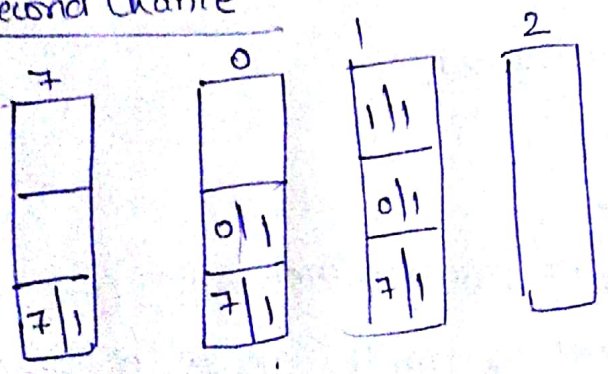
Q Calculate no. of page faults.

FIFO → 15
optimal → 9
LRU → 12
} Done,

Second chance → 11
LFU → 13
MFU → 14,

Now calculate for second chance algorithm.

Second Chance



Page replacement, Frame allocation } → to implement virtual memory

Frame allocation: How no. of frames are allocated to a process to increase degree of multiprogramming. Atleast some minimum no. of pages be allocated to each process to avoid too many page faults,

9/4/2018

Frame Allocation Algorithms.

① Equal Allocation

If we have m frames and n processes, then we allocate m/n frames to one process.
Allocate equal frames to all process.

Problem:

Large process require more frame and small processes require less frames. Hence not good algo

② Proportional Allocation.

He we allocate no. of frames to a process proportional to a size of its virtual memory (size of its address space)

Large process → more frames.

Local page replacement.

In case of page fault in page replacement, one page of process was replaced by another page of same process.

(Page replacement within the process)

Global page replacement

In case of page fault, we bring a page from disk & replace any page of any process in memory.
One page of 1 process can replace other page of other process.

Local

→ easy to implement
→ Process has more control over its execution

Global

→ throughput performance is better
→ process will lose its autonomy
→ more commonly used.

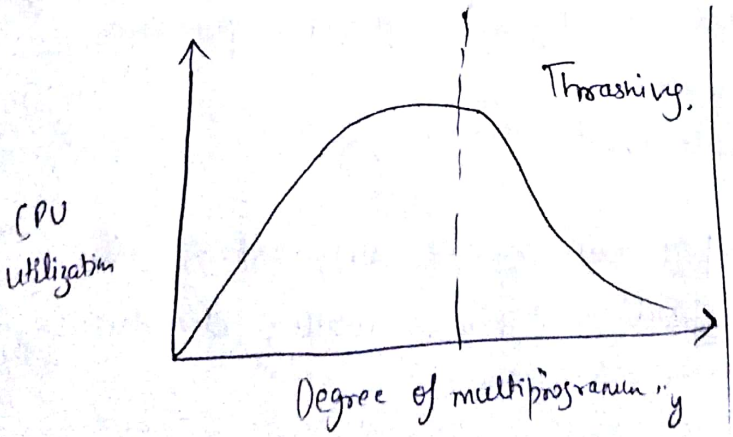
Thrashing (Short note on this)

→ Taking more time in servicing page fault, but actual processing time taken is less.
→ It does not good ~~when this happens~~.

When this happens?

→ If a process has less frames, page fault very frequently., spend time in page fault & no useful work happening in system & CPU utilization go down.

Another module, longe term scheduler.
System performance is increasing, hence its increasing degree of multiprogramming.
So its loads some processes in memory. But frames increases even more and more page fault. Then its, induces more processes & chain reaction occurs.
this is thrashing.



First CPU utilisation increase, then starts saturating and beyond this further, performance increases and thrashing occurs
CPU utilisation diminishes due to high page fault

→ Thrashing is not a good thing.

→ You have to try to avoid this.

→ Computer Engineers have found out two techniques to avoid threshing

① Working Set Model

→ Based on locality of reference.

→ A process is accessing a particular set of pages again at current point of time is working set.

Working set of a process i =

→ WS_i = { pages referenced in the last k accesses } (10/20/5... any)

(size of $P_i \leq k$)

→ $D = \sum_i |WS_i|$

m = no. of frames in the memory.

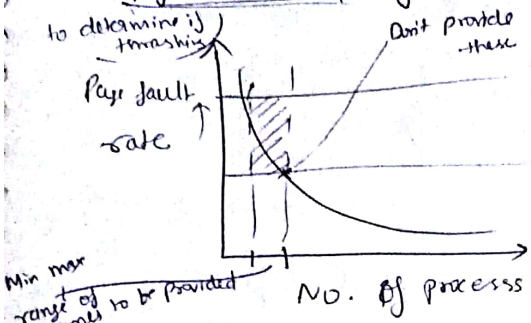
If $D > m$, then threshing.

① should not try to increase degree of multiprogramming don't push processes, ② try to swap out processes from memory ③

to avoid threshing

Maintain & keeping track of working set of all processes is difficult. Hence another disadvantage

② Page Fault Frequency.



Here we define two threshold levels.

→ upper bound

→ lower bound

We need to ensure that page fault b/w upper or lower bound or below upper bound.

→ If we decreases page fault below lower bound, then lots of frames to a process, degree of multiprogramming decreasing and CPU utilisation goes down.

→ Above upper bound, thrashing occurs.

→ Try to ensure no. of frames within the box,

Here ends, Virtual management.

two units left → By the end of this week (syllabus complete)

10/4/2019

STORAGE Management (Management of secondary storage devices)

It is something different from Memory Management (Difference — size of RAM is much smaller than size of disk.

RAM is volatile and disk is permanent

Hard disk are the most common forms of secondary stay. (Magneto electronic device

→ It is not a purely electronic device. (if it is, needs of capacitors to store, and will lose info when power turned off)

→ Operating system abstracts the physical properties of the disk.

→ Operating system interprets the bits stored in the magnetic files.

→ Files are the named collection.
→ collection of related information.

→ Files exist only on hard disk, floppy drives

→ File is the smallest ~~allocation~~ allotment of logical data.

→ File can store both code and data.

→ All files have their formats.

→ OS imposes file system on the hard disk.

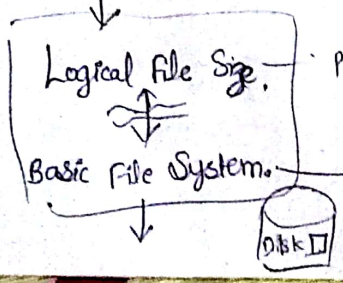
→ Using a file system, it is easy to store files, retrieve files and search files.

Two issues?

→ ~~how~~ how the users see files? ← user perspective

→ how the files stored on disk? (break-hard disk, mapped on the disk? no file physically)

Application Program (say Ms word:) 'Kavith.docx.



provide path of file (it is interested to read/write

decides where the file actually stored in the disk

Job of file system to map file to disk.

There is something called directory.
Now they are called folders (in windows)

There are three techniques for storing file in a disk.

→ 1) Continguous Allocation
2) Linked Allocation,
3) Indexed Allocation.

Here store diff. parts of file at diff. location and have a contents like in a book.

First 500 bytes are at this location

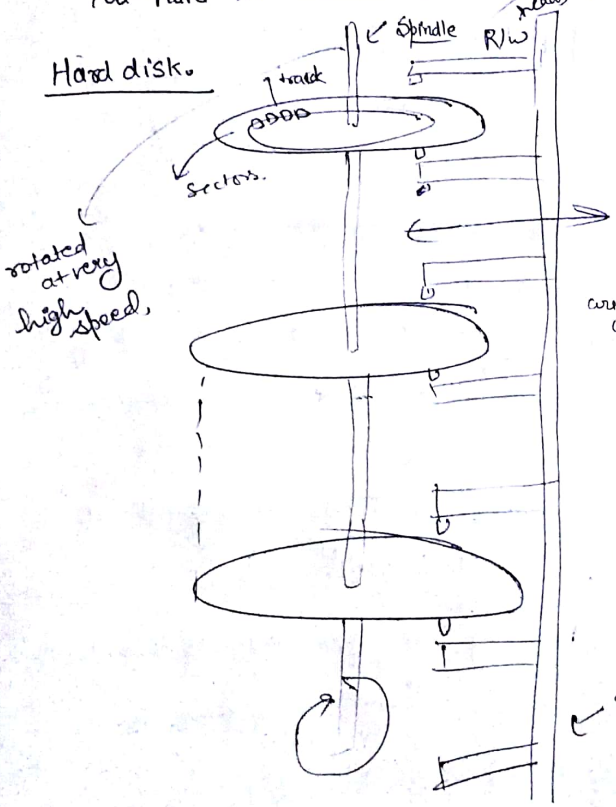
next bytes are at other location

force ?
or

Free Space Management

You have to maintain a linked list of free ~~frames~~ spaces.

Hard disk.



rotated at very high speed,

The whole devices are called hard disks

Multiple metallic platter are called, Platters.

Over that, one layer of plastic.

These platters are of same size, same length.

Arm Assembly

→ R/w heads can be placed over any position of the disks.

→ These R/w heads and Arm assembly move together.

→ This is a circle, how will you stored the bits?

→ A track is a imaginary circle

→ All tracks have sectors.

→ We write bits in the sectors.

→ Set of all such tracks which can be accessed for one particular position of the arm assembly is called cylinder (that can be accessed by a particular position of r/w head)

= 2 × no. of platters.

no. of bottom surface

→ Cylinders are logical entity, not physical.

→ More than 1000 cylinders in disk.

→ We have to define some parameters for performance

① Rotational Latency. (Mostly hardware independent)

It is time taken by r/w head to reach on the top of required sector.

② Seek Time

to move the arm assembly so that r/w head reach the required cylinder or track. (Rectilinear motion)

→ H/w dependant (cannot change).

→ (It can be minimised by the operating system using s/w

(what are diff. addresses of hard disk?)

→ We tried to store the files in continuous sectors.

specified by sector no.

also track no.

when all tracks are full,
then we move to another cylinder.

→ So, we also store,

Cylinder no,

Cylinders should be numbers from 0 to max.
inner - n_{max}
outer - 0

* 0th cylinder is at the periphery of the cylinder.

In numerical

There are two ways of storing bits:-

① Uniform bit density. ⟷
Distance is same b/w every bit.

② Non-uniform bit density.
⟷ at corner
⟷ at center.

There is something
called (block)
↙
is the unit of memory which is read or write
in one go, or in one operation.

Using software, we can play with seek time, we can improve seek time of disk.
how?

① Disk Scheduling.

Different programs to access diff. cylinders. → You have one queue to access different cylinders.

Objective → to minimize seek time.

People have developed some algorithms.

Called disk scheduled Algorithms

Disk Scheduling Algorithms

~~Hard disk~~ ⇒ we receive request to perform input/output on following cylinder.

Ex→ Queue = 98, 183, 37, 122, 14, 124, 65, 67

No. of cylinders = 200

Curr. pos of R/W head = 53.

We have do find seek time.

but we find, no. of cylinders traversed by R/W heads

(1) FCFS

First come First Service.

No. of cylinders,

FCFS → 640 cylinders.

> In exam, first we have to draw trace diagram.
> Trace of R/W head

(2) Shortest-seek-time-first (SSTF)

SSTF → 236 cylinders

SJF, Optimal. not possible

This is possible to implement

(3) Scan algo ≡ Elevator Algo.

Read write heads move from 0 to 199 then 199 to 0,

Whichever request it gets, it executes it.

Total number of cylinders = 331

In this, we have to specify current direction.

* Consider this in exam

* Inward → 0 – 199
  Outward → 199 – 0

④ LOOK algo

U turn from max,

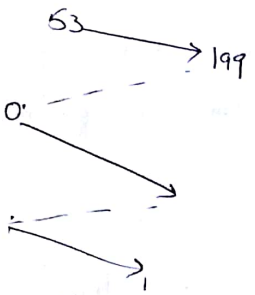
Look algorithm is smarter from **Scan** algorithm

⑤ Circular-SCAN ≡ C-SCAN.

It will go to 199,

then return to 0

but in return journey it will not service anyone.



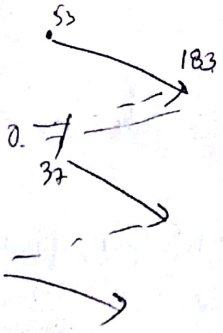
It will look like it is solving the issues in circular manner,

⑥ C-LOOK Algorithm

It is just like the CSLAN,

but here it will not move to 199,

but it will ~~not~~ move to 183 only.



12/4/18

Disk Formatting.

(Manufacturer formats the disk and store OS and other software into it)

Hard Disk → main storage device.

Without formatting the disk, you can store anything in this.

Formatting
↳ two types

① Physical Formatting ≡ low level formatting. (takes long long time)

Whenever there is a new disk, you haves to perform this.

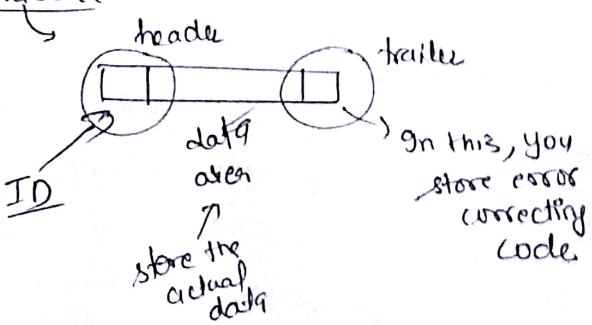
After formatting

In this, first of all you have to fill all the space (platters) with special data structures. (filled in each of these sectors).

(Once you done this, your disk is ready to use)

When you perform this, then only tracks, cylinders are created.

Data Structure



Now the OS can store the files in the data area,

Manufacturer selling ~~data~~ Hard disk, first have to do physical formatting

(2) Logical formatting.

→ In logical formatting, you do not create sectors.

→ You just create a new file system.

(It formates all the data)

Block → that amount of data to and from from the disk in one operation.

OS allow some application programs to perform raw I/O. (I/O without the help of file system)

This will make things faster.

Bad Block → If logically formatted a disk, only a part of floppy disk was problem and rest is fine

we used to use floppy disk, (magnetic disk)

those areas on which sector has become defected.

these blocks on which we can not perform read/write operation.

DIR (LS) : Total memory and bad blocks

linux

Some of the bad block can be rectified by logical formatting

(That is all about storage management)

(read about file management structure)

(Short note on operation of disks)

Practice → Numericals

Disk scheduling algos

System Protection and Security

(Short notes)

Protection and security

They are related but not synonyms.

Protection

From whom?

↳ from genuine and internal errors

(Some files have overwritten other files.)

or

Process does not read OS model)

Security

↳ from intention external threads.

→ OS generally takes care of protection., do not provide security.

OS allows or cooperate with specialised programs which provides security.

Ex- Antivirus..

Protection

→ Any hardware or software ~~object~~ resource is called object

→ Collection of/set of permission, → domain

Process can be in one domain

Process P_i in one domain can have only those permissions.

Access matrix

	O1	O2	O3	O4
D1	R			
D2		RW.		
D3				

→ Any process of type D1 can perform Read operation on the resources of type O1.

→ Any process of type D2 can perform Read/Write operation on the resources of type O2

There are three types of accesses.

→ (1) Read Access (R)

→ (2) Write Access. (W) can change the resource

→ (3) Executable (E)

	O1	O2	O3	O4
D1	R			R
D2		RW	R	RW
D3			E	

Let O3 is an executable file.
Process in D3 can execute the file in O3.

→ A process may require to change its domain.
(If is allowed.)

If is also shown in ~~adjancy~~ Access matrix.

S stands for switch.
Any process in D2 can switch to domain D1.

	O1	O2	O3	O4	D1	D2	D3	
D1	R			R				
D2			R	RW*	S		S	
D3	R		E		S	S		

(*) Any process in type D2 can allow permission to ~~other~~ other process to access resources of type O4.

→ copy permission

↳ can be of three types.

① transfer

after allowing, itself will not be allowed to do access

② Copy limited.

Let P_7 ↑ P_{10}

Child process,

P_{10} can access resources but cannot grant permission to other process

(further copy not allowed)

③ copy unlimited

P_7 ↑ P_{10} ↑ P_{30} ↑ P_{30}.

All information shown in the matrix
Called
↳ access matrix *

very large matrix
(but it also very sparse one
(Most of the cells are empty)

(Storing sparse matrix is always tricky)
(You are wasting a lot of memory)

Better way to store access matrix,

↳ You can store it like this,

For each non-empty entry.

< domain, object, rights > (as triplets)

< D1, O1, R >
< D2, O2, RW* >

Still this will be a long one.

Other methods

① ~~Access lists of objects~~

For each object, we will be having access list. (same as column of access list but leaving blank spaces)

Ex: O1 ⇒ < D1, R >, < D3 R >

② Capability list of domains.

Here you have one capability list of each domain. (Equivalent to one row of access matrix)

&

D3 ⇒ <O1, R>, <O3, E>, <D1, S>, <D2, S>

Just this, for system protection.

Go through this.

(Short note on this)

(In book, recursive (extra) :-)

Security

We require to secure against external and intentional threats.

Threats → are proven techniques through which our computer can be damage. which stealing the data.

nowadays, also include, (illegal access with malicious intend)

Most talked threat is

① Computer virus.

(is infamous threat)

What is a virus?

a computer program

Dangerous property → It is self replicating

And it is infectious

(capability to infect more and more parts)

What it will do?

↳ It will modify or destroy your file.

→ These computer viruses can not exist alone.

They are code fragments.
(they attach with different files)

Viruses are spread through Internet, COs, pen drives, floppy disk.

Viruses are of different types

→ macro viruses
affect Ms office files.
(stay as a code of macro).

→ stealth,
→ Multipartite
→ —
→ —
→ —

Read from book (Shortnote)

Study of computer virus is known as computer virology.

② Trojan horses

↳ (Things look good from outside but may not from inside)
files. (look attractive that has harmless and attractive cover (like Lottery))
but when we click, it affect our computer
(copies virus into our system)

They are generally used as virus carries. They will transfer virus to system

⑤ Worm

→ are standalone programs/files (Do not need other programs)

Unlike Virus.

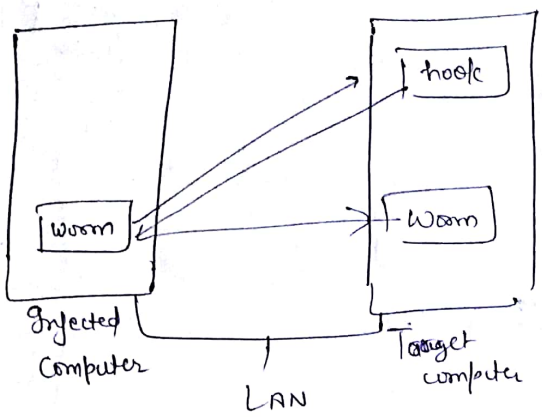
→ generally. spread through Local area network

—Worms can go through lab to another.

— → spread through networks

infectious programs

In book, diagram



→ Worm wants to effect to target computer

→ Worm creates a hook

→ Hook sents requests for copying the worm,

→ Once worm reaches the system, it look for other on the n/w.

→ Each computer knows the address of other comp. in system

More threats are there, in the book (read from book) may not increase your marks but increase your knowledge) ☺

Antivirus / Antimalware try to delete worm/trojan horse

Firewall

- , are not meant to neutralised threats.
- → to maintain the flow of information → what is downloading, what is uploading..
- → for security
- → stop you from accessing certain sites, or ~~set~~ connecting to some LAN.

Firewall can do,
both good or bad thing.

Ex→ File,
CV cannot sent through mail.

Encryption
Digital Structure
read
from
yourself

Ends