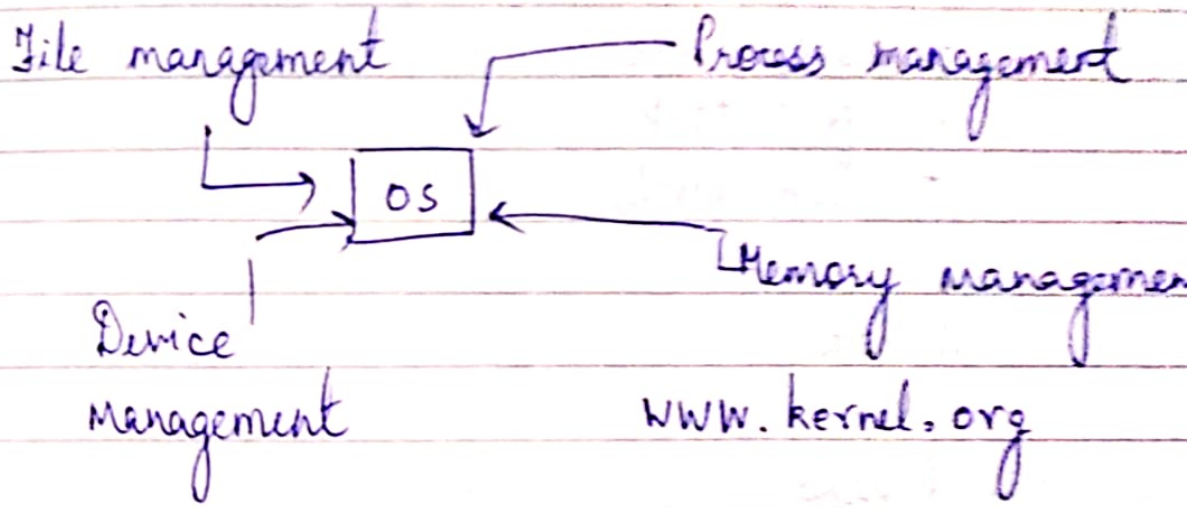
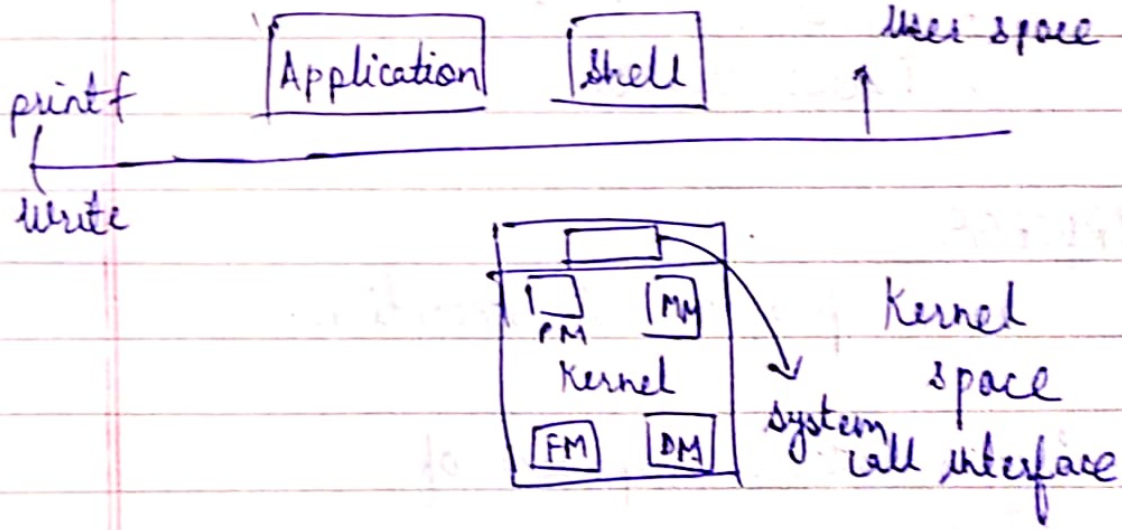
Page No.:
Date: / /

Grading policy (Relative)

Midterm - 30%
Endterm - 40%
Project - 20%
Quiz 2 - 10%

OS is a program which acts as an interface b/w hardware & user of the computer.





Linux OS architecture

- Linux booting
- Linux OS Stack

- GRUB
- Boot image
- INIT task
- Run levels

Agenda

- Process
  - Process context
  - Process life cycle
  - Process creation

PROCESS

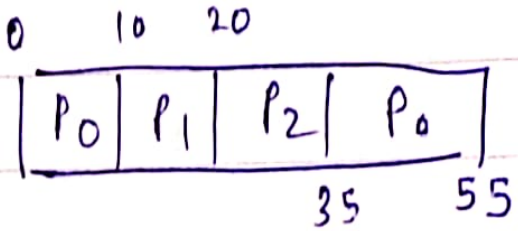
is a program in execution.

- Process context comprised of
  - Text
  - Data
  - Stack
  - Other resources

- Preemptive SJN
- Preemptive priority scheduler
- Round Robin
- Multilevel queue
- Multilevel feedback queue

Preemptive SJN

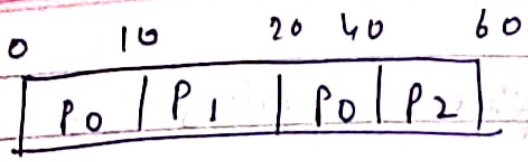
Pr	Service Time	Arrival time
P_0	30	00:00
P_1	10	00:10
P_2	15	00:15

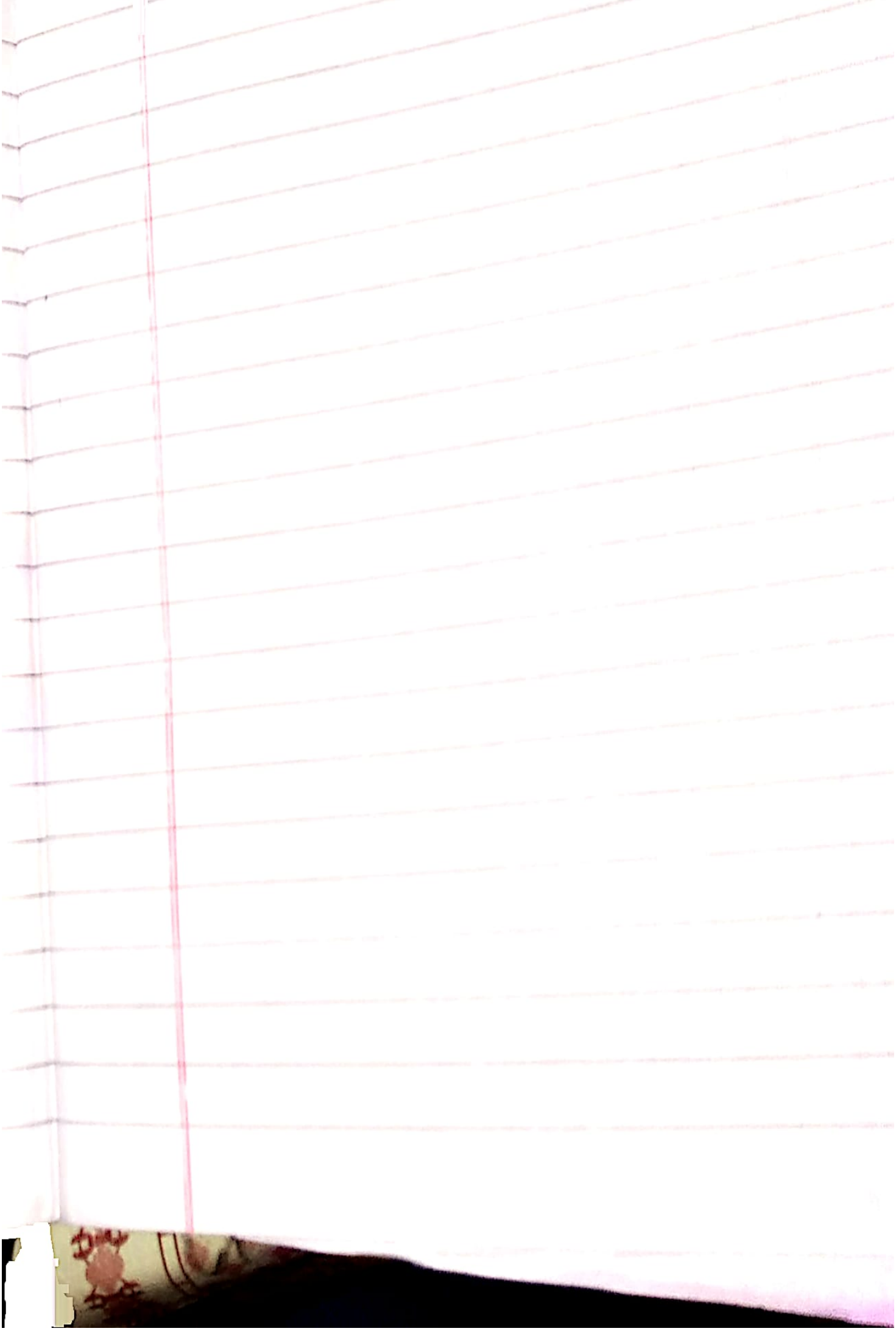


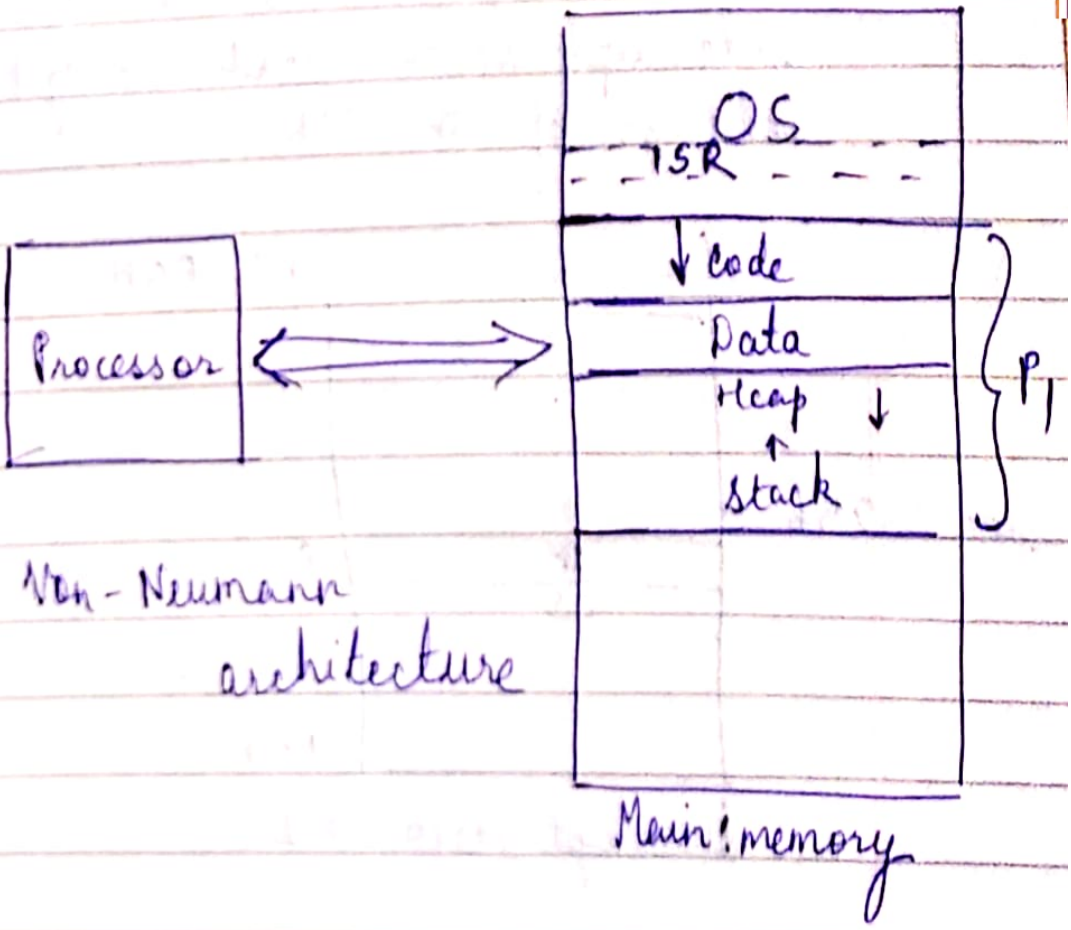
Preemptive priority scheduler

Pr	Service time	Priority	Arrival time
P_0	30	2	00:00
P_1	10	1	00:10
P_2	20	3	00:15

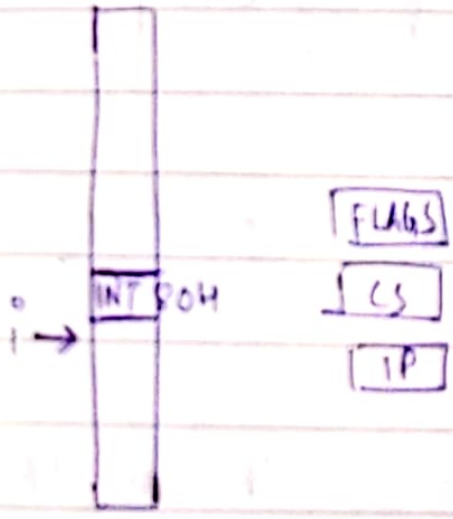
Page No. :
Date / /





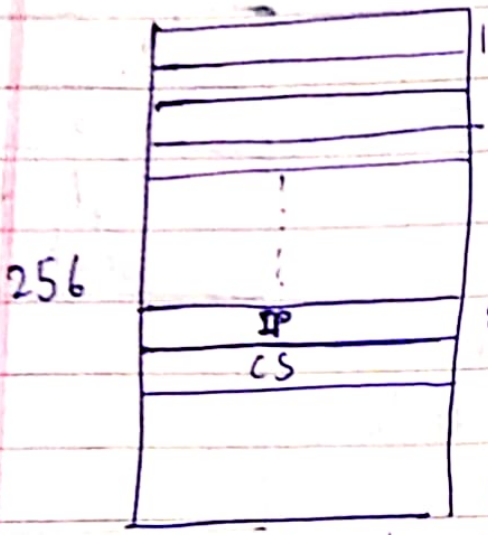


Von-Neumann architecture



When INT occurs, FLAGS, CS & IP is stored on the stack.
(done by the machine (os))

When interrupt occurs, the OS refers the interrupt vector table to get the starting address of the ISR.



Interrupt vector table #

First 4 bytes are for interrupt vector no. 00
next 4 " " " 01H
:

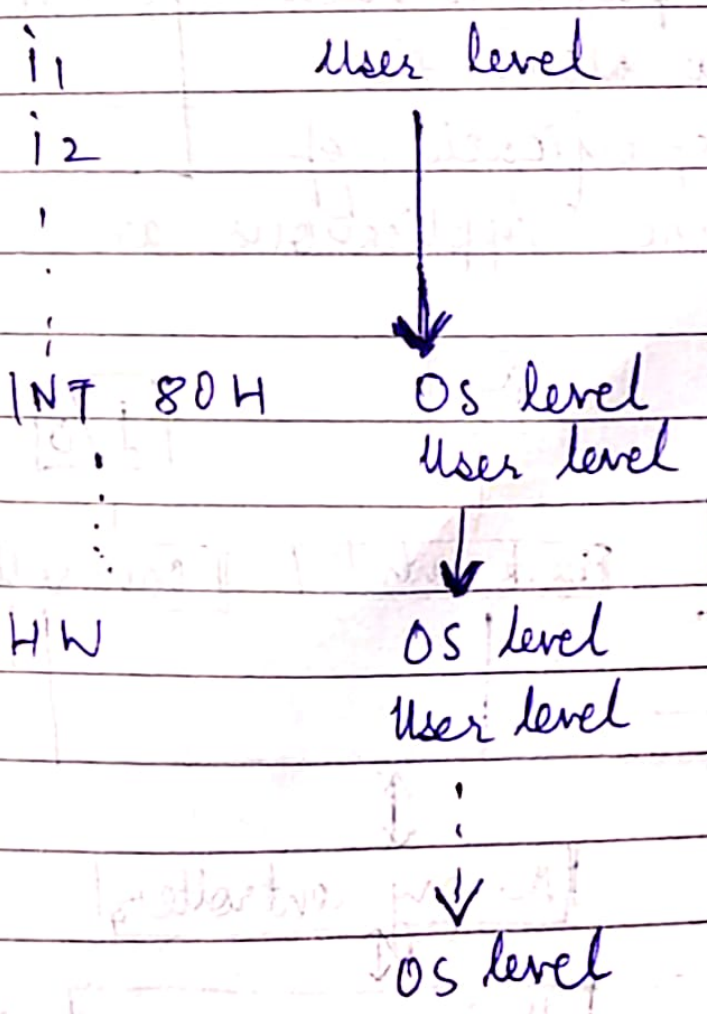
Total 1kB of data is stored in interrupt vector table.

→ The last instruction in ISR is IRET.
IP, CS & FLAGS are restored from the stack.

We also need to restore the register values. This can be done in the program by PUSHA before the interrupt instruction.

IDTR contains the starting location of the interrupt vector table. The OS designer can decide where the interrupt vector table should be stored in the OS.

PROCESS

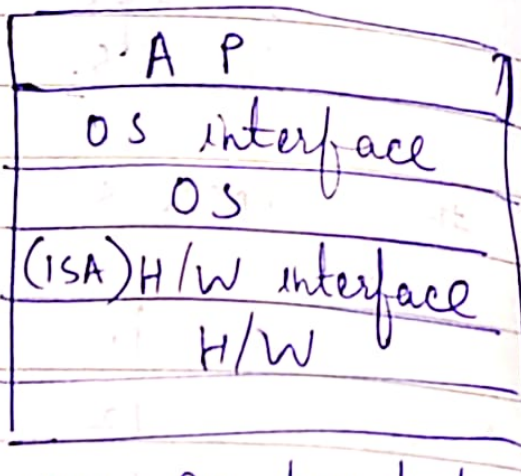


Page No.:
Date 11

80X86 -4

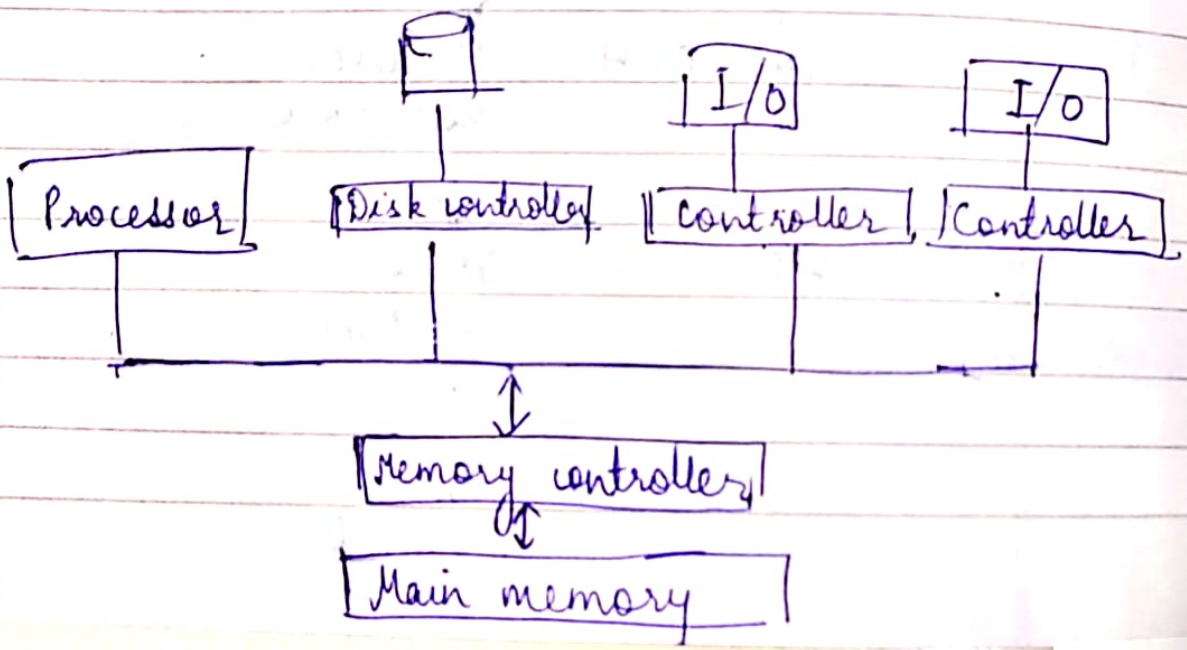
Ring level

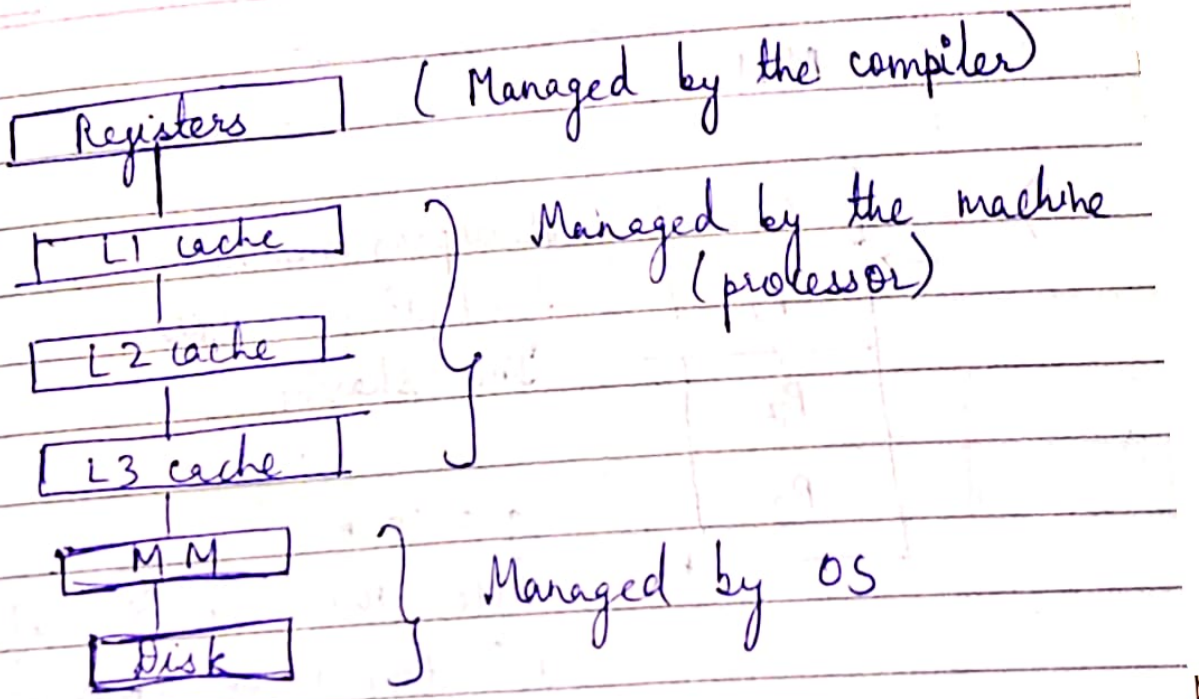
00 ← OS (Highest priority)
01
10
11

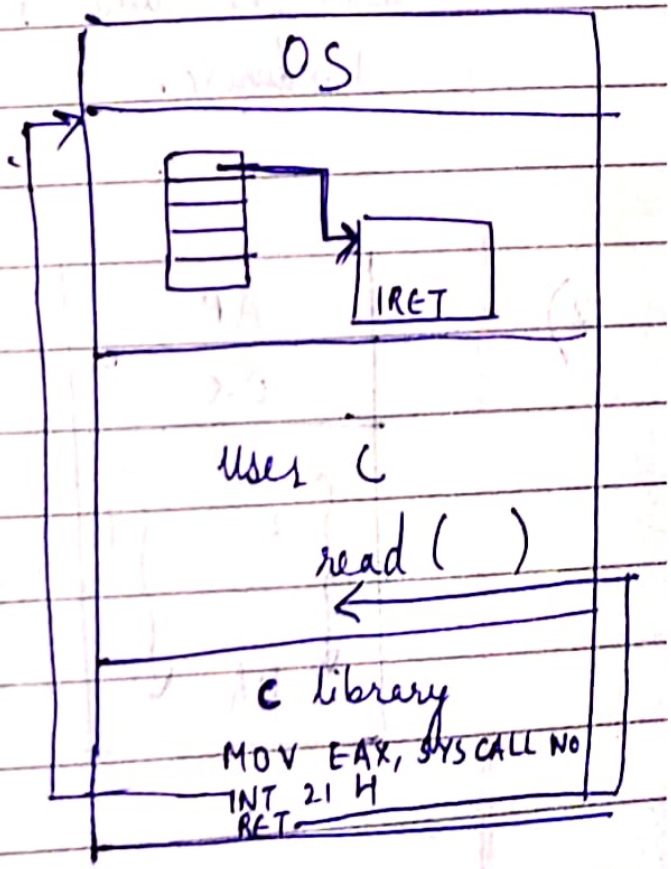


(System calls)

Application programs need not know about the hardware specification of the machine. Applications are OS dependent.

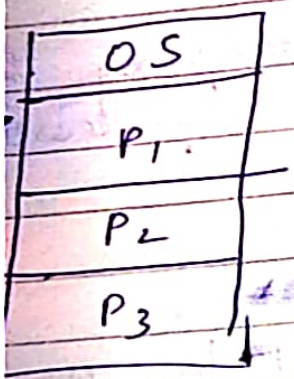






Page No.:
Date / /

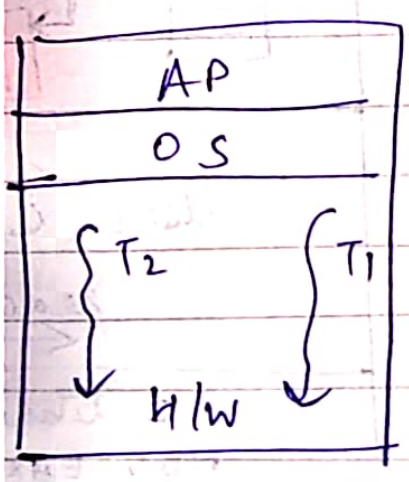
MM



Multiprogramming
Multiple processes
Time sharing

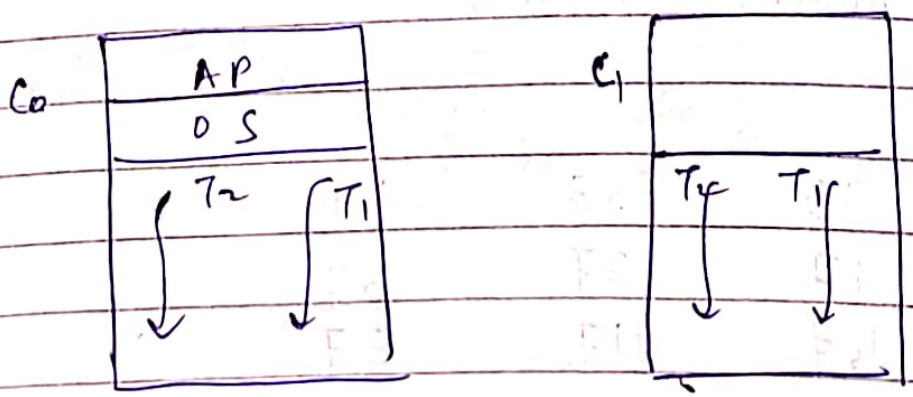
q = 10 ms
After 10 ms execution of process is stopped & execution of next process is started.

on The time is measured by a timer in hardware.



Two threads ⇒ 2 PC values, 2 regist in the same core
Two processes run at a time.

3) Multi-core architecture



4 processes at a time.

fork() :- This system call creates a new process.
Returns the process id.

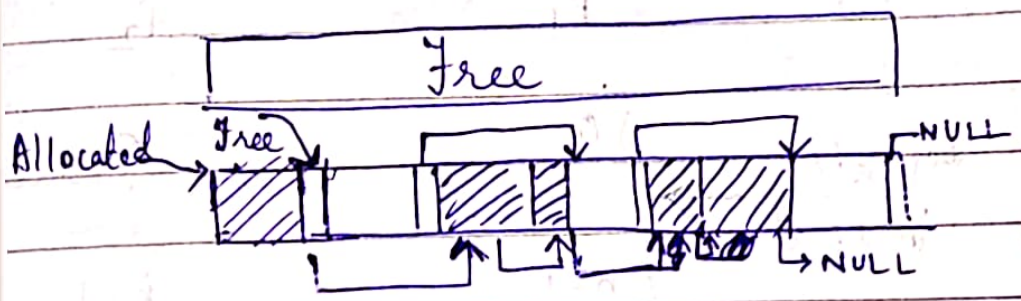
```
int main()
{
    pid_t pid1, pid2;
    n = 10;
    n = n + 5;
    pid1 = fork();
    if (pid1 == 0)
    {
        n = n + 20;
        printf("%d", n);
    }
```

Page No.:
Date / /

```
    n = n + 2;
}   printf("%d", n);
```

35	35	17
17	37	35
37	17	37

MEMORY MANAGEMENT

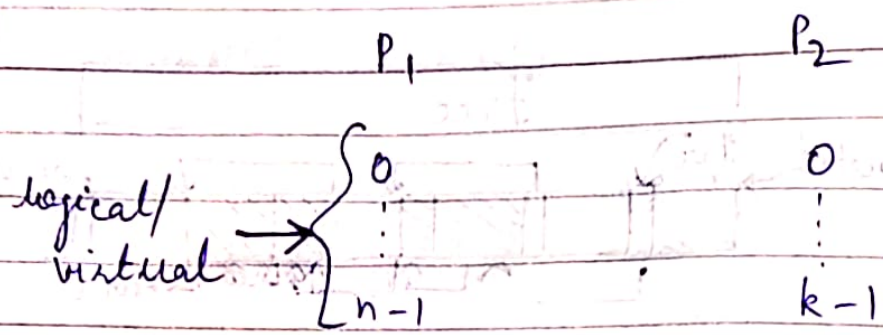


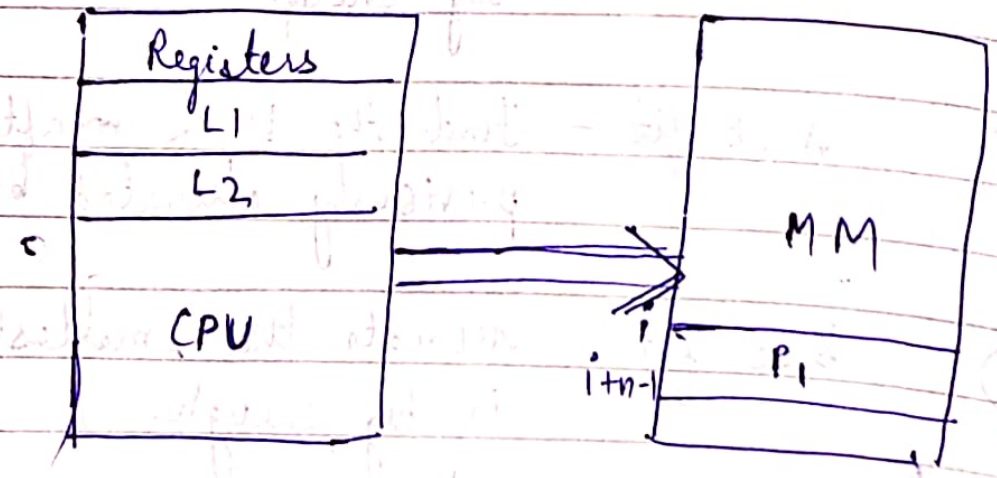
1) First fit - Find the first block which is big enough

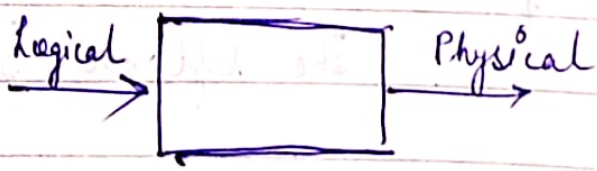
2) Next fit - Find the block after the previously allocated block.

3) Best fit - Allocate the smallest block which is big enough

4) Worst fit - Allocate the largest block, The leftover block is still big.







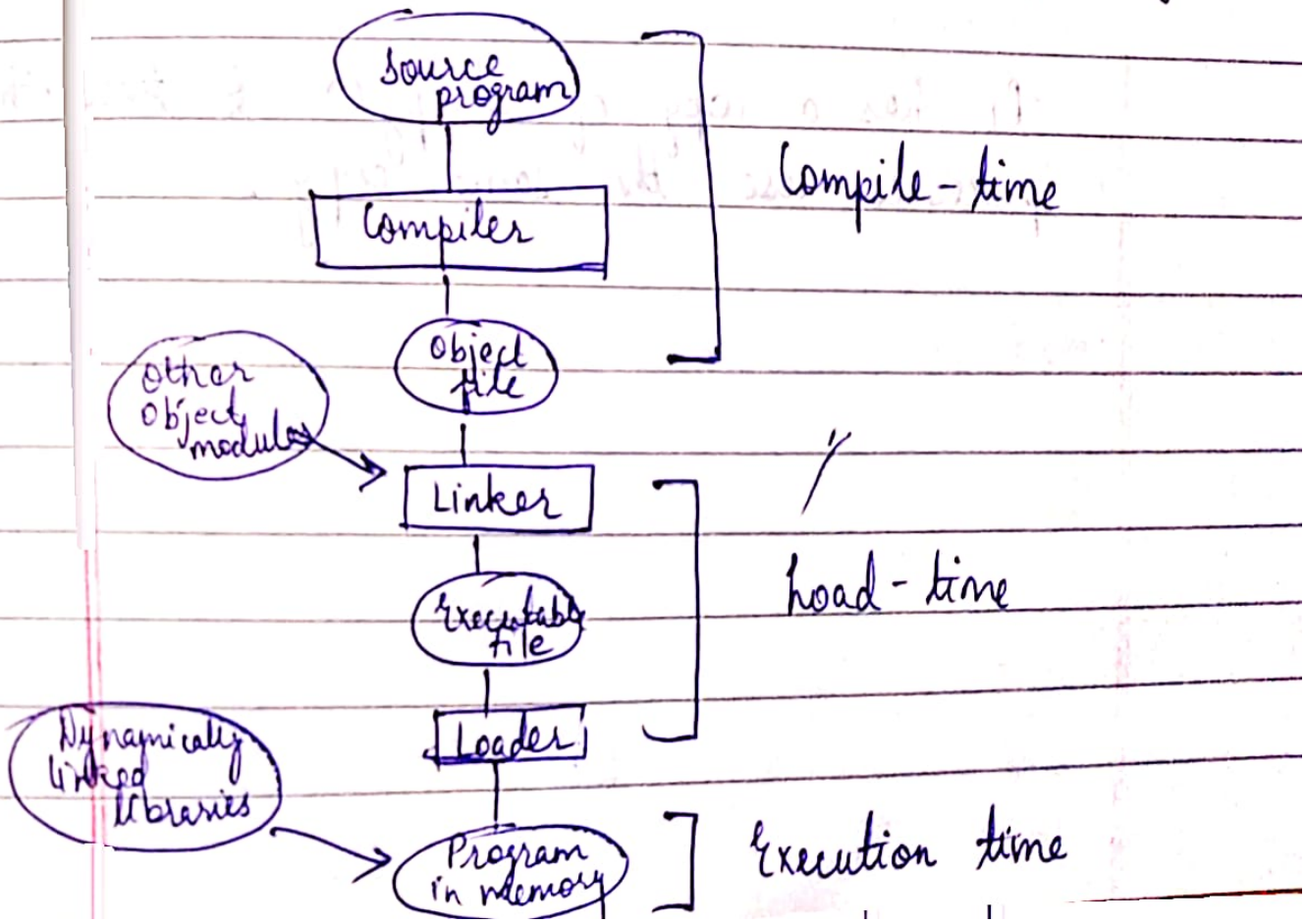
MMU
(Memory management unit)

Page No.:
Date: / /

Compile-time binding - The physical addresses of the processes are fixed at compile-time so it is not possible to load it anywhere else in the MM afterwards.

Load-time binding - loader loads the program into the MM at load-time.
It is not possible to move to another location at execution time.

Execution-time binding - At run-time the translation of addresses takes place (logical → physical).



required. The module have to be already linked.

DYNAMIC LINKING

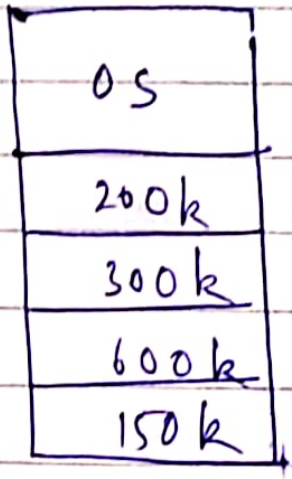
10 programs → strcpy()

In dynamic linking, only one copy of strcpy() is present & can be used by all the processes.

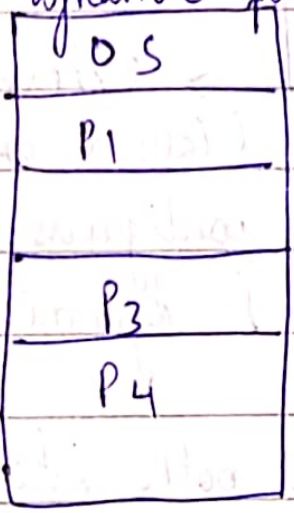
P_1 has a copy of strcpy() & then other processes use the same copy.

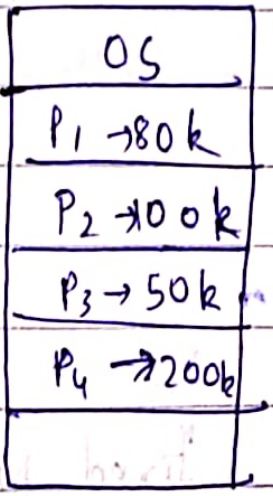
PARTITIONED MEMORY ALLOCATION (CONTIGUOUS ALLOCATION)

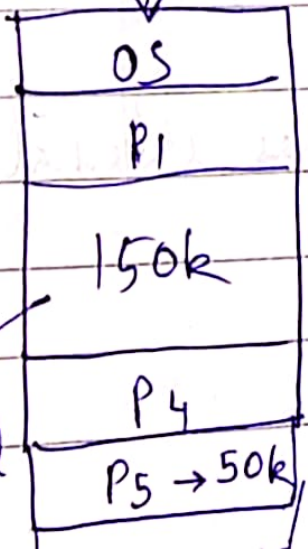
→ Fixed partition - Cannot change size during run-time.



→ Variable / dynamic partition



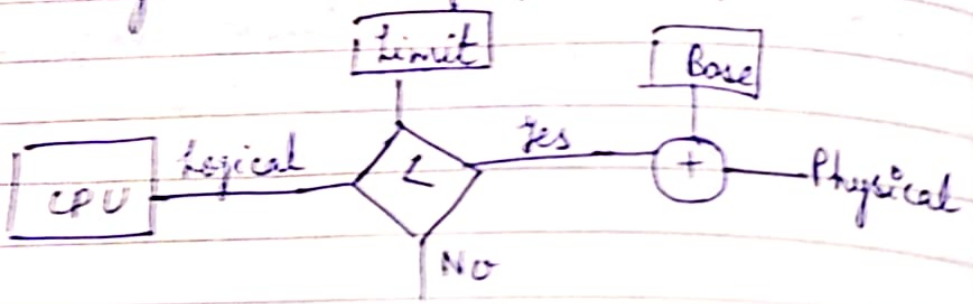




P5 comes into MM

Processes completed

Base register - starting location of process
Limit register - Size of process.



≡ FRAGMENTATION → Internal (Due to fixed partition)
→ External
(Caused due to when b sufficient contiguous memory is not present in dynamic allocation)

Fixed partition has both internal as well as external fragmentation

Dynamic partition has external frag.

→ External fragmentation can be resolved by compaction. Compaction is a costly operation. The processes have to be reallocated into one used area & remaining area becomes free.

→ Avoid contiguous allocation to resolve external fragmentation

This can be done by paging.

PAGED MEMORY (Modern computers use this)

Logical address space

P1

Page 0
Page 1
Page 2
Page 3
Page 4

Page size = frame size

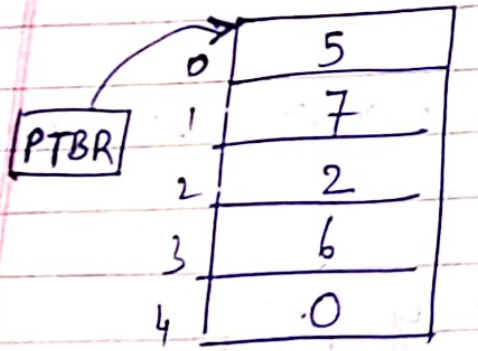
Main memory

0	Page 4
1	
2	Page 2
3	
4	
5	Page 0
6	Page 3
7	Page 1

Divided into frames

Page table stores where the pages are loaded in the main memory. Each process has a page table

Page table



Index of page table is the page no.

E.g

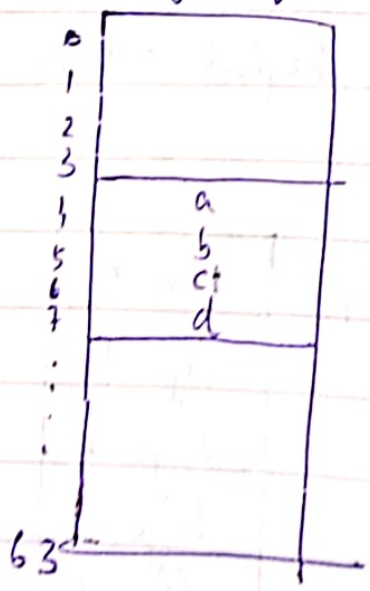
$2^6 = 64$

Physical address = | f (4) | d (2) |

f → Frame no.

d → offset

64 bytes of data

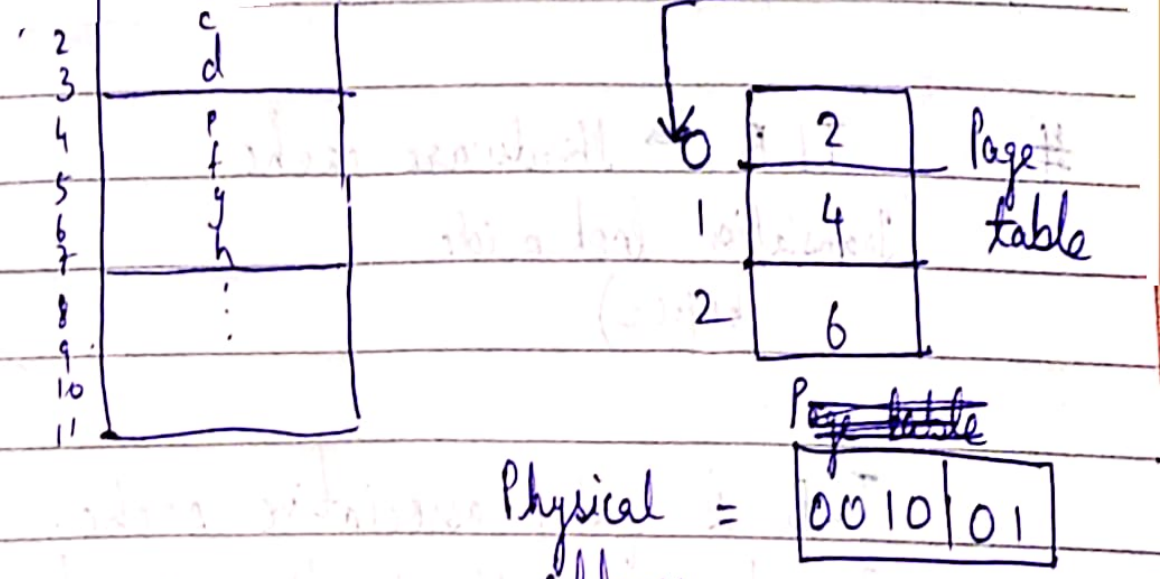


Memory frame

Logical address = | p (4) | d (2) |

p → Page no.

d → offset



→ There can be internal fragmentation in case of paged memory.
If process requires 10kB & page size is 4kB, 2kB is wasted.

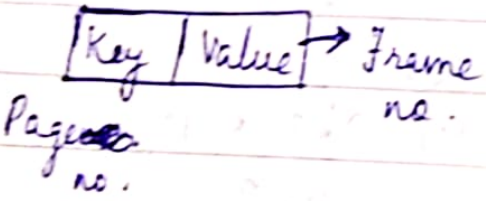
→ If page size = 4kB
Max. internal frag = 4kB - 1
Min " " = 0
Avg. " " = 2kB

There is no external fragmentation as any free frame can be allocated.

Code pages can be shared among processes.

TLB → Hardware cache
(Translation look aside buffer)

TLB is fully associative cache.
The key of all of them can be compared together.



Page No.:
Date: / /