Department: Computer Science & Engineering

Program: BSc. in CSE

Course No: CSE 3213

Course Title: Operating System

Examination: Semester Final

Semester (Session): Spring 2020

Student ID: 170204070

Signature & Date: 24/05/2022

Student ID: 17020970
Course NO: CSE 3213
Signature and Date: 24.05.21

Ans to the Question no - 02

2 (a)

a) Answer: CPU utilization means how effecientlly Operating system use the resource of Computer and make sure that CPU remain idle in minimum time.

CPU ~~utilit~~ Utilization = $1 - p^n$ [where n = no of process, p = process waiting for I/O (In this case cpu remain idle)]

For the given scenario,

Pseudo code for Reader -

```
Reader () {
While (True) {
        down (mutex);
        readerCount ++;
        If (readerCount == 1)
                down (db);
        UP (mutex);
        Reader Reading ();
        down (mutex);
                readerCount --;
                if (readerCount == 0)
                        UP (db);
        UP (mutex);
}
```

Student ID: 1202040720
Course No: CSE 3213
Signature and date: 24.05.21

~~Here~~ ~~mutex~~ ~~Se~~

Pseudo code for Writer –

```
Writer() {
while (True) {
        Prepare Data for Write();
        down (db);
        Write Data()
        UP(db);
}
}
```

Here, mutex ~~use for~~ binary semaphore use for lock ~~taken~~ unlock when new reader arrival or reader leave.

for reader,
at first a new reader first time arrive, he down the mutex so that no other readers can not able to increment or decrement shared variable reader count. then he increment the reader variable. If he is the first reader he have to lock the full system so that writer can't interrupt when a reader is reading, for this I use another binary semaphore 'db', then reader UP mutex so that other new reader can come and read. then ~~reader~~ after reading reader leave the system so, he again lock to decrement shared

Variable readerCount. and then check, If he is the last reader or readerCount == 0 that means after his leaving no other reader in the system so ~~that~~ he for that he up(db); now new writer can come and write.

when a writer enter the system he lock db by down(db); so that no other reader or writer cannot enter the system. then writer write his data and unlock the system by up(db); so, new writer or reader can enter the system.

—

2(b)

b) Answer:- **Dispatcher Module**: the dispatcher Module is the module that gives control of the cpu to the process selected by scheduler.

~~The~~ function of dispatcher module -

(i) Switch context
(ii) Switch to user mode
(iii) Jump to the proper location in the user program to restart that program.

Student ID: 1202040R0
Course No: CSE 3213
Signature and Date: 24.05.21

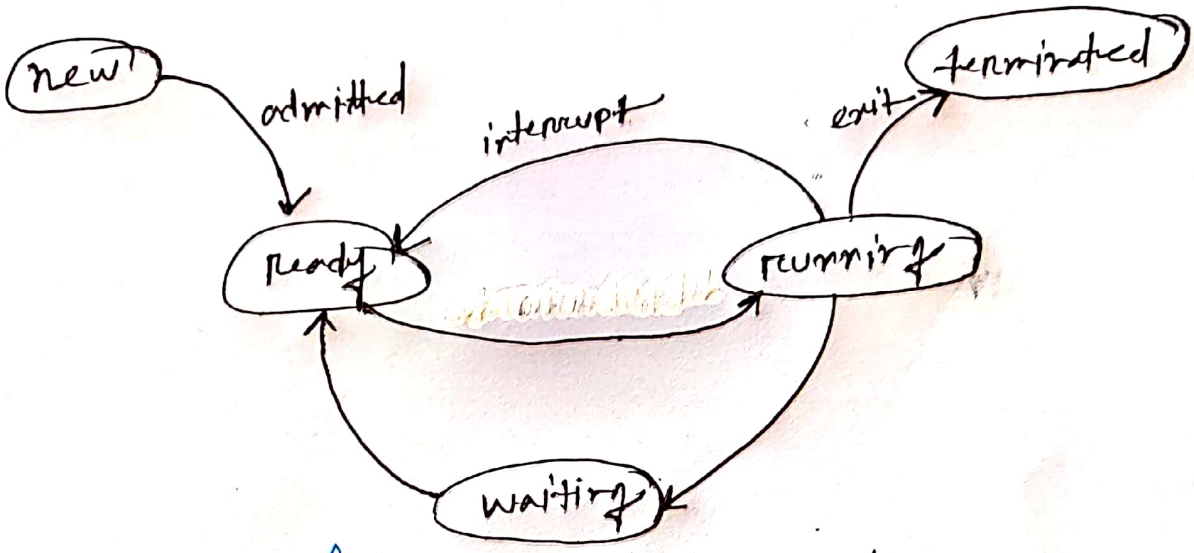


fig: Diagram of process state

When a process created it is called new process then, after created process when its instructions are executed is called running state. before running the process have to wait for the ~~equal~~ CPU being assigned called ready state. It ~~take~~ wait for some event like I/O from user. for this the process is in waiting state. Firally when the process finished execution, it is called terminated.

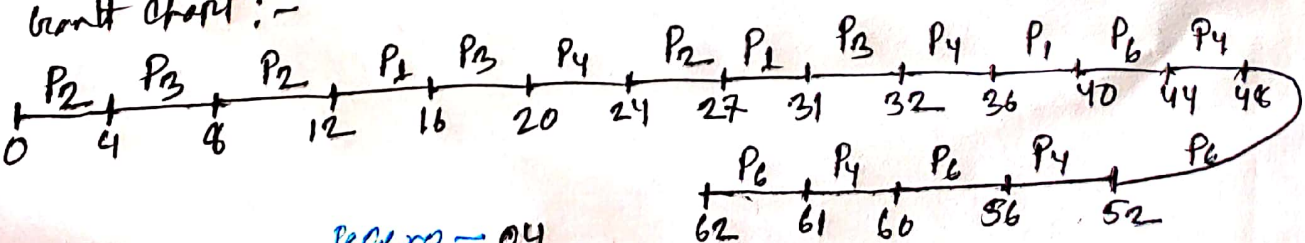
2(c)

c) Answer:

Round Robin :-

given time quantum = 4

Gantt chart :-



Student ID : 1702090 70
Course No : CSE 3213
Signature and Date : 24.05.21

Waiting time :

$P_1 = (12-5) + (27-16) + (36-31) = 23$

$P_2 = 0 + (8-4) + (24-12) = 16$

$P_3 = (4-4) + (16-8) + (31-20) = 19$

$P_4 = (20-12) + (32-24) + (44-36) + (52-48) + (60-56)$
$= 32$

$P_6 = (40-35) + (48-44) + (52-48) + (56-52) + (61-60)$

$= 14$

∴ Average waiting time for RR $= \left(\frac{23+16+19+32+14}{5}\right)$

$= 20.80$

Response time :

$P_1 = (12-5) = 7$

$P_2 = 0$

$P_3 = 0$

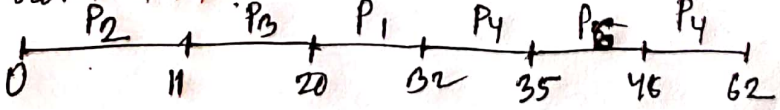
$P_4 = (20-12) = 8$

$P_6 = (40-35) = 5$

∴ Average response time $= \left(\frac{7+8+5}{5}\right) = 4$

SRTN Scheduling :

Gantt chart :



Waiting time : $P_1 = (20-5) = 15$

$P_2 = (0-0) = 0$ $P_3 = (11-4) = 7$

$P_4 = (32-12) + (46-35) = 33$

$P_6 = (35-35) = 0$

∴ Average waiting time = $\left(\frac{15+0+7+33+0}{5}\right)$

= 11

~~Average~~ response time, $P_1 = (20-5) = 15$

$P_2 = (0-0) = 0$ $P_3 = (11-4) = 7$

$P_4 = (32-12) = 20$

$P_6 = (35-35) = 0$

∴ Average response time = $\left(\frac{15+7+20}{5}\right)$

= 8.4

Comparing the RR & SRTN, we see that for this 5 given processes Average waiting time for RR (20.80) is greater than average waiting time for SRTN(11). but RR has faster response time. the response time for RR(9) < SRTN(8.4).

Student ID: 170204070
Course No : CSE 3213
Signature and date : 24.05.21

Ans to the Question no 5

5(a)

5) a) Answer: As we all know that page fault occurs when a program attempts to access a block of memory that is not stored in the physical memory or RAM. So, when CPU gives a memory request is first checked in MMU to make sure it was a valid memory request. If the reference is invalid, OS detect it as page fault.

Given that,

Page frame size = 16 KB

RAM Size = 16 MB

Address space size of the process = 32 MB

So, No of frame $= \frac{16\,MB}{16\,KB} = \frac{16 \times 2^{20}}{16 \times 2^{10}} = 1024$

No of Page $= \frac{32\,MB}{16\,KB} = \frac{32 \times 2^{20}}{16 \times 2^{10}} = 2048$

Offset Size $= 16 \times 2^{10} = 16384$

A process request data from 200th page, offset 15200

So, the logical address will be,

Page no in 11 bit binary & offset in 14 bit binary.

Logical address/ Virtual address = $(00110010000)_2$ (page no) | $(11101101100000)_2$ (offset)

Page no – 7

Student ID : 17020407020
Course No : CSE 3213
Signature and date : 24.05.21

Physical address, data available in 28th memory Frame
Frame no will be 10 bit
Offset will be 14 bit

Physical Memory address = | $(0000011100)_2$ | $(11101101100000)_2$ |
frame no | offset

5(b)

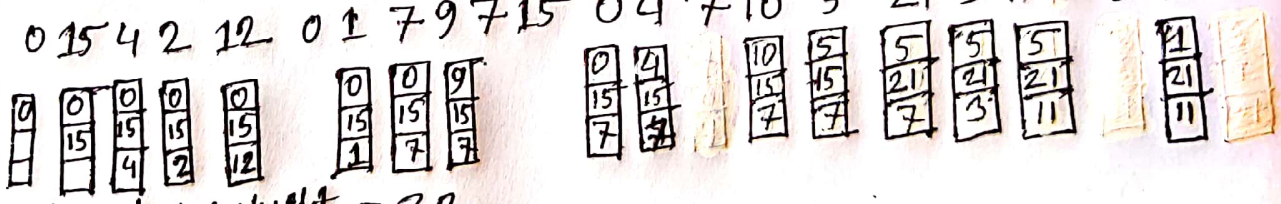
5(b) Answer:

Here,
Given that frame no = 3

So, For Optimal

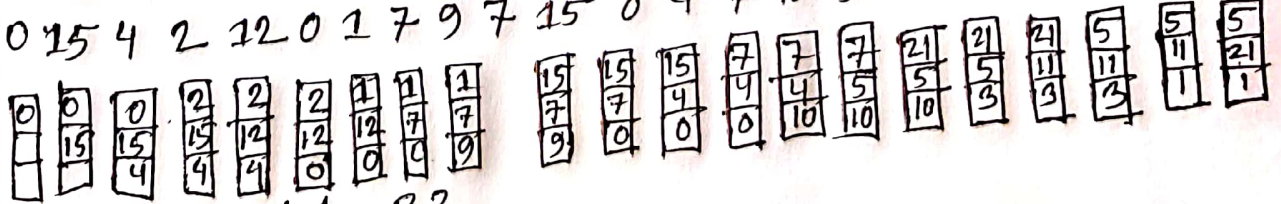
~~0 15 4 2 12 0 1 7 9 7 15 0 4 7 10 5 21 3 11 5~~

0 15 4 2 12 0 1 7 9 7 15 0 4 7 10 5 21 3 11 5 1 21



total request = 22
Page fault for optimal = 16

0 15 4 2 12 0 1 7 9 7 15 0 4 7 10 5 21 3 11 5 1 21



total request = 22
Page fault for LRU is 21.

Student ID : 170204070
Course No : CSE 3213
Signature and Date : 24.05.21

5 (c)

c) Answer :

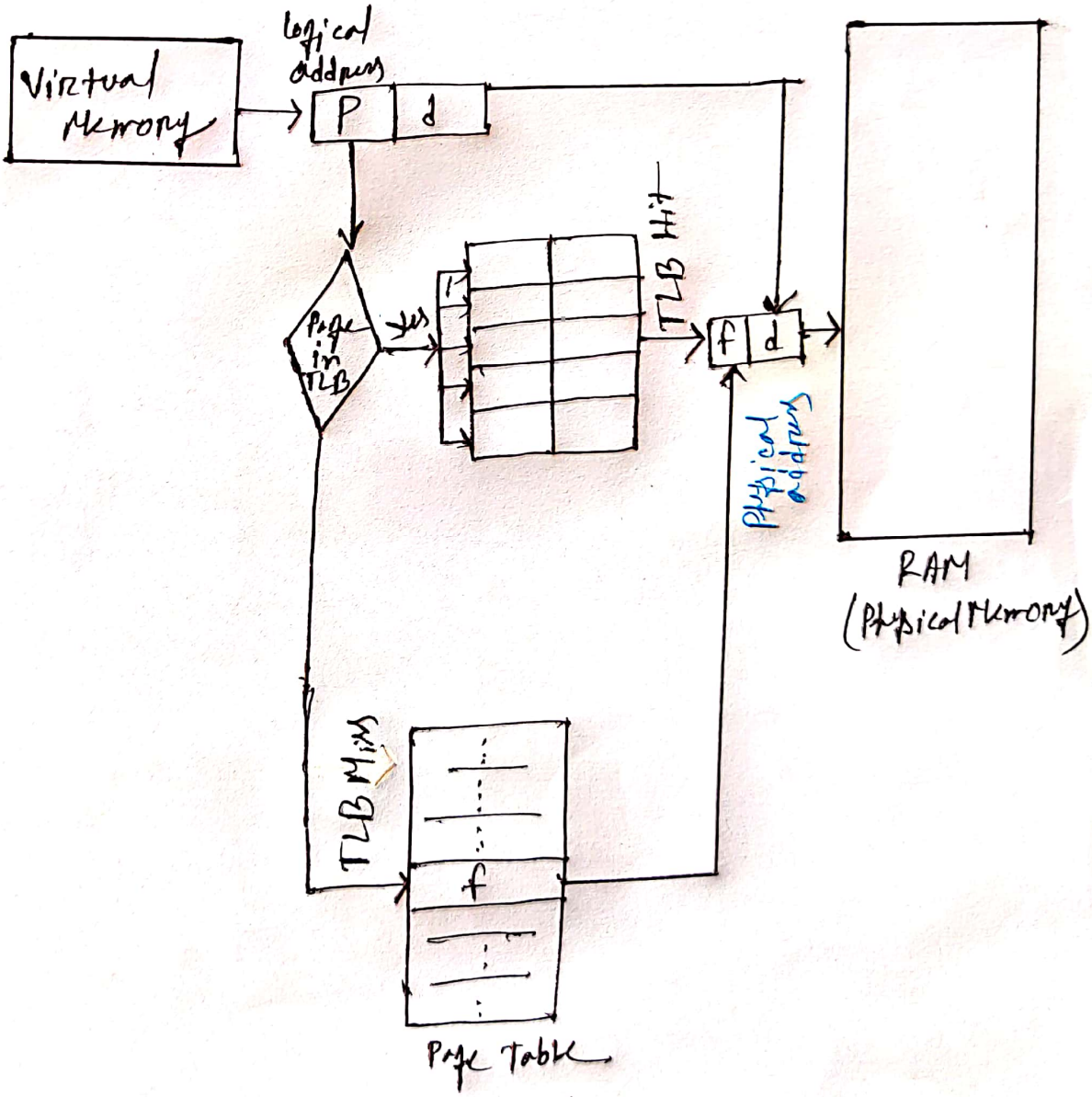


fig: Address translation Process.

here, first time Virtual memory / CPU generate data. ~~form~~ for this, give page no and address as logical address. Then, the page no search in page no - 9

TLB. If the page found in TLB then it is called TLB hit and give frame no and translate to physical address and then take data from physical memory.

If the page not found in TLB called TLB miss. Then search data from page table then make physical address. If the page not found in page table then it is called page fault and search page from physical memory then load it into page table.

Student ID : 170204070
Course No : CSE 3213
Signature and date : 24.05.21

Ans to the Question no - 4

4 (a)

a) $E_{n+1} = \alpha t_n + (1-\alpha) E_n$

where,

E_{n+1} = Estimated cpu time for process at $n+1$

E_n = Estimated cpu time for process at n

t_n = Actual cpu time taken by a process at n

$$E_{n+1} = \alpha t_n + (1-\alpha) E_n$$

$$= \alpha t_n + (1-\alpha)\{\alpha t_{n-1} + (1-\alpha) E_{n-1}\}$$

$$= \alpha t_n + (1-\alpha)\{\alpha t_{n-1} + (1-\alpha)^2 E_{n-1}$$

$$= \alpha t_n + (1-\alpha)\cdot \alpha t_{n-1} + (1-\alpha)^2 \{\alpha t_{n-2} + (1-\alpha) E_{n-2}\}$$

$$E_n = \alpha t_n + (1-\alpha)\cdot \alpha t_{n-1} + (1-\alpha)^2 \alpha t_{n-2} + (1-\alpha)^3 \alpha t_{n-3} + \cdots \cdots + (1-\alpha)^{n+1} E_0$$

Now,
Put $\alpha = 0$, means

$$E_{n+1} = 0 \times t_n + E_n$$

$\therefore E_{n+1} = E_n$ that means always same estimation

Student ID: 170204020
Course No: CSE 3213
Signature and Date: 24.05.21

$\alpha = 1$; $E_{n+1} = t_n$ that means Previous Actual cpu time is next estimation

$\alpha = 0.5$ means

$E_{n+1} = 0.5\, t_n + 0.5 E_n$

the average of Previous Actual cpu time & estimated cpu time is our next Estimation.

—

When there is any Saved variable or resource is shared into different process then race condition arise.

Suppose, we assume a Printer daemon program. there is a queue where many process puts their data for printing and since the resource are shared, multiple process can put data in the same slot. If context switching occurs before accessing the in, out counters, there race condition arise.

Student ID: 170204070
Course No: CSE 3213
Signature and Date: 24.05.21

4(b)

b) To prevent race condition there are some method/ technique, one is mutual exclusion. That means One process can enter or use shared variable at a time.

1. ~~Mutual Exclusion~~
2. ~~No preemption~~
3. ~~Hold & wait~~

Another approach is no process can block other process from non critical region.

No assumption about speeds or no of cpus.

No starvation - no process waits forever to enter critical region.

there are some algorithm where trying to solve race condition. they are, Peterson solution, TSL, XCHG. But there are busy waiting problem. This busy waiting problem decrease the cpu utilization so this approach is not preferable.

Student ID : 170204020
Course No : CSE 3213
Signature and Date : 24.05.21

Q(b)

A semaphore is a variable or abstract data type used to control access to a common resource / shared variable by multiple process and avoid Critical Section problem. ~~There are two types~~ A Semaphore is integer type variable and its value cannot be negative.

There are two types of semaphore

1: Binary semaphore

2. Counting semaphore.

Binary semaphore value only 0, 1.

Student ID: 170204070
Course No: CSE321B
Signature and Date: 24.05.21

4(c)

c) Answer: The instruction save in the disk, but not in execution/running state is called program.

The program in execution mode it's called process.

When a program load in ram and then execute there are 3 segment allocate for the program, they are Data segment, Stack segment, text segment. Then we called that program as a process. Every process have unique process id, and other details saved in PCB.

Student ID: 1202040?0
Course No: CSE 3213
Signature and Date: 24.05.21

Peterson

Peterson is good approch. ~~in busy waiti~~ But there is a problem of busy waiting. As we all know that busy waiting decrese the utilization of cpu. So that can't be a good solution for avoiding race condition.

Strict alternation

When there is two process one is fosten then the another process then in strict alternation, the slower process can block the fosten process while in non critical reagon. So, slower process outside the critical region block fosten process

Student ID: 1702040x0
Course No: CSE 3213
Signature and date: 24.05.21

Ans no - 7

7(a)

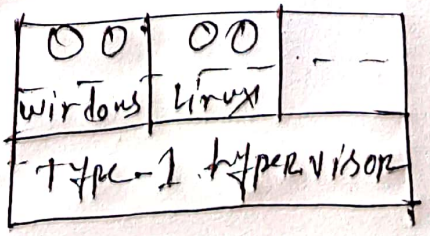
a) Ans: Thread creation is preferable instead of replicating the whole process because, for replicating process they are two different (although work same). they have different processil, need space for code, data, registers files, and stack.
but to ~~creat~~ create a new thread just need to have different registers and stack only. no need of space for code data and files. Same space so, thread creation is preferable instead of replicating process.

Student ID: 1202040 70
Course No: CSE 3213
Signature and Date: 24.05.21

7(b)

b) Answer: Type - 1 hypervisors run directly on the system hardware. They are installed directly on physical host server. Hardware just rely on operating system and provides higher performance, validity and security. It runs in dedicated hardware. They require management ~~control~~ console and used in data centers.

example: Oracle, OVM for SPARC, KVM etc.



Student ID: 170204070
Course No: CSE 3213
Signature and Date: 24.05.21

Type-2 hypervisor run on a host Operating system. when the ~~virtualization~~ Virtualization movement first began, to take off, the were most popular Ex: Oracle virtual box, Solaris zones.

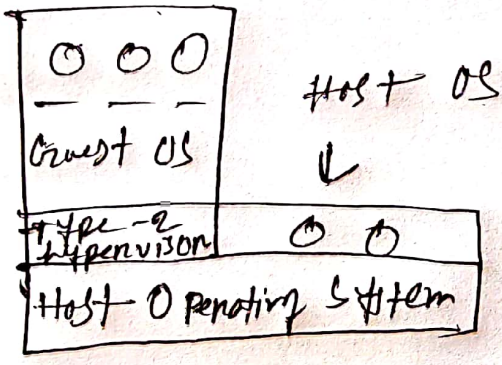


fig: Type-2 Hypervisor

7(c)

c) To recover process from deadlock OS need to take some recovery approach. They are —

1. Resource Preemption
   - Roll back
   - Victim Solution

for roll back there need a checkpoint

for back.

Then taking resource from another process. The process resource taken from process called victim selection.

2. Process kill
   - all processes in deadlock
   - gradually kill process

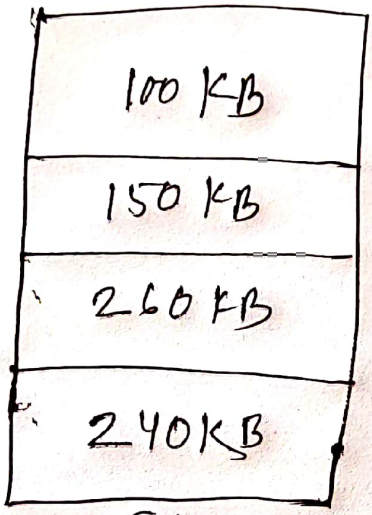
When all process in dead lock, we need to gradually kill process. for gradually select and kill case, same process can kill every time. So, the process can be in starvation. So, we need to have a track that how many times a Process kill to get chance.

Student ID : 170204070
Course No : CSE 3213
Signature and Date : 24.05.21

Ans no - 06

6(a)

6(a)

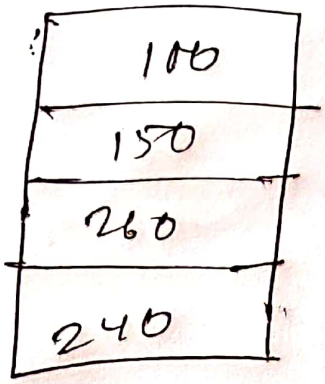


(a)

request 30 KB, 90 KB, 100 KB, 80 KB, 260 KB.

Now, if we give 260 KB slot to the 30 KB, there remains 230 KB (using worst fit). then 90 KB from 240 KB, ~~250~~ 150 will remain, then 100 from 150 KB slot remain 50 KB, then 80 KB from 150 remain have 70 KB. Now we can't give slot to memory but we have 450 KB available. that is called external fragmentation. and the remaing ein every slot is internal fragmentation.

Student ID: 170204070
Course No: CSE 3213
Signature and Date: 24.05.21

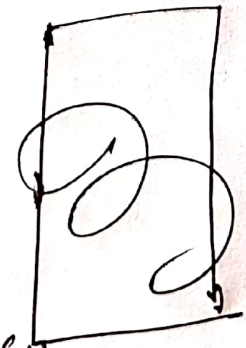


request : 30, 90, 100, 80, 260

for best fit :

30 will give to 100 slot → remain 70
90 will give 150 slot → remain 60
100 will give 240 slot → remain 140
80 will give from 140 remain → remain 60
260 will give from 260 → remain 0

So, we satisfy all request there are no external fragmentation



for same data,

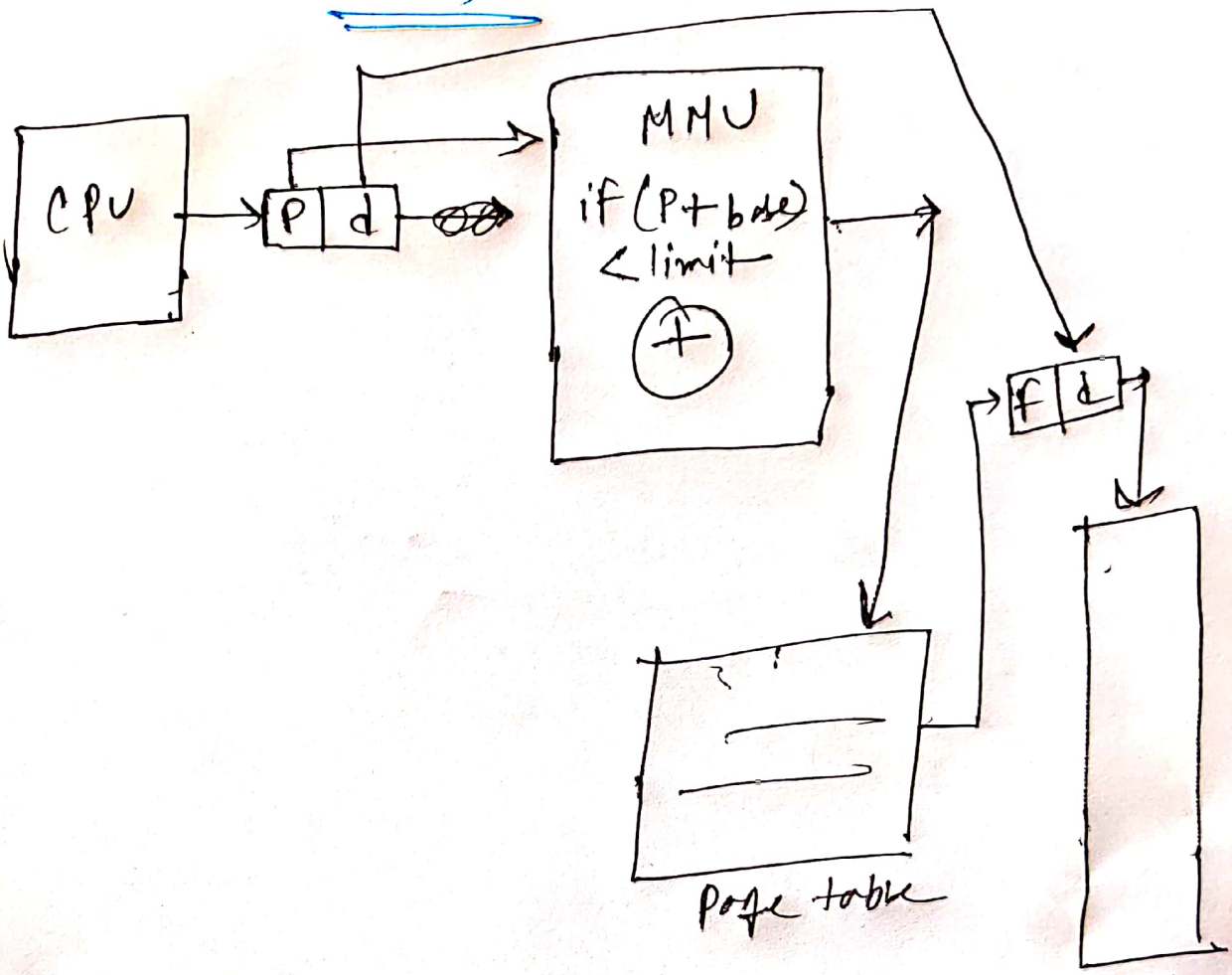
worst fit :

30 from 260 → remain 230
90 from 240 → " 150
100 from ~~180~~ 230 → " 130
80 from 180 → " 100

260 request can not accept. So, there are 450 external fragmentation.

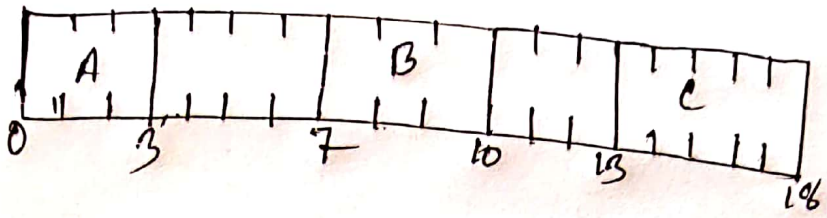
Student ID : 170204070
Course No : CSE 3213
Signature and Date : 24.05.21

6 (b)



in contigeous allocation if P+base < limit then MMU add the ~~cat~~ base to limit and send them to physical ram. as they are Contigeous allocation so, NO process can go other process area. Physical RAM.

Student ID: 170204070
Course No: CSE 3213
Signature and Date: 24.05.21



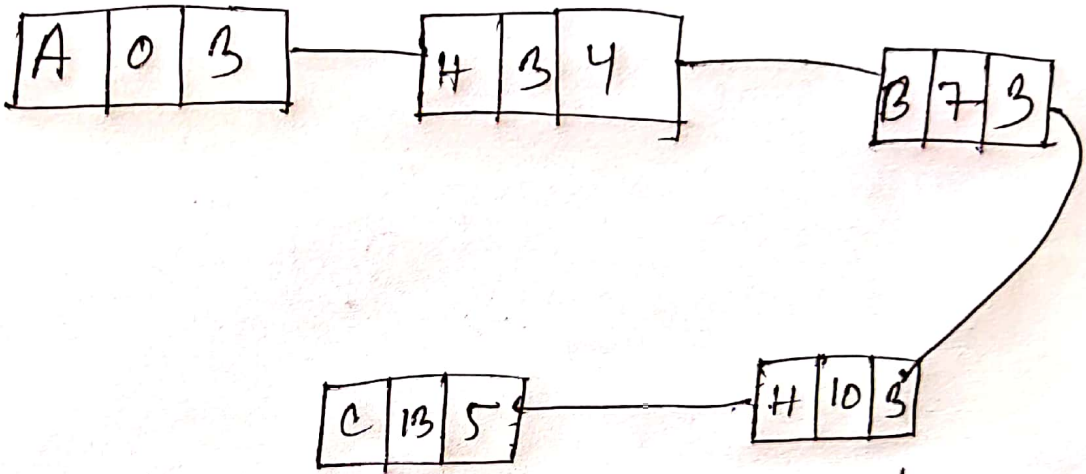


fig: Hacking code using linked list

6 (c)

c) The size of physical memory is limited. So, if the program size/ process size is greater than the physical memory then we cannot load the program in the physical memory. for this we need virtual memory to load the complete program.