Department: CSE

Program: BSc in CSE

Course No: CSE3213

Course Title: operating system

Examination: Semester Final

Semester (Session): Fall2020

Student No: 18.01.04.072

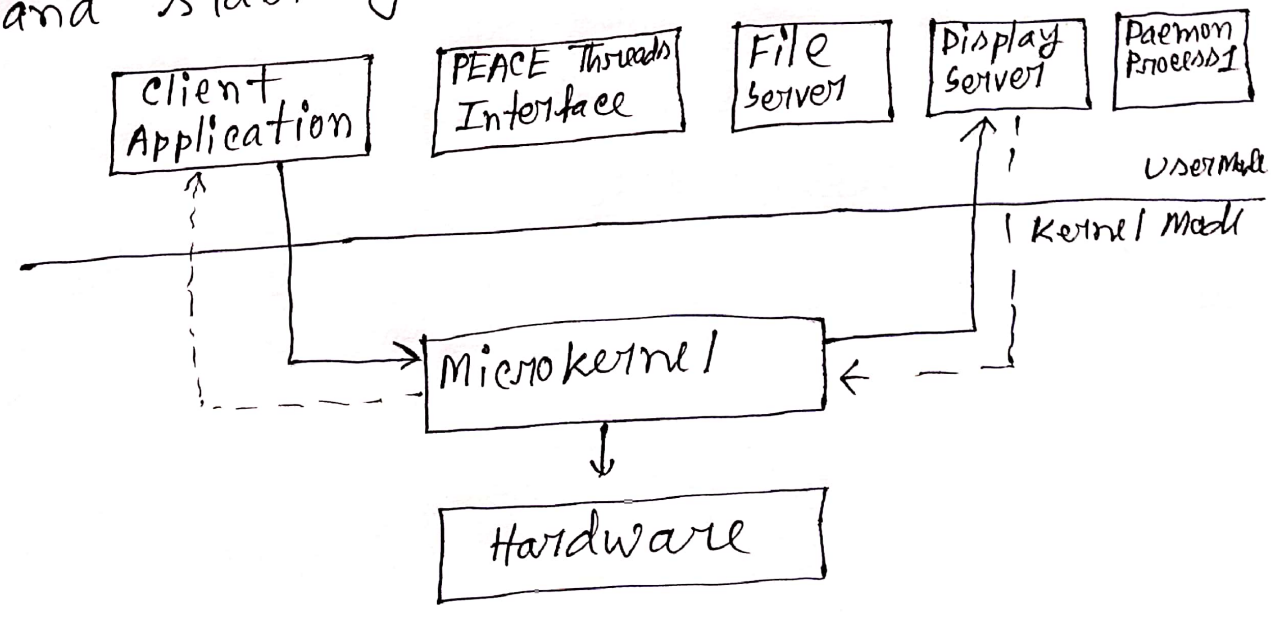
Signature and Date:
Rakib, 24october,2021

Student ID: 18.01.04.072
Course No: CSE3213 Rakib

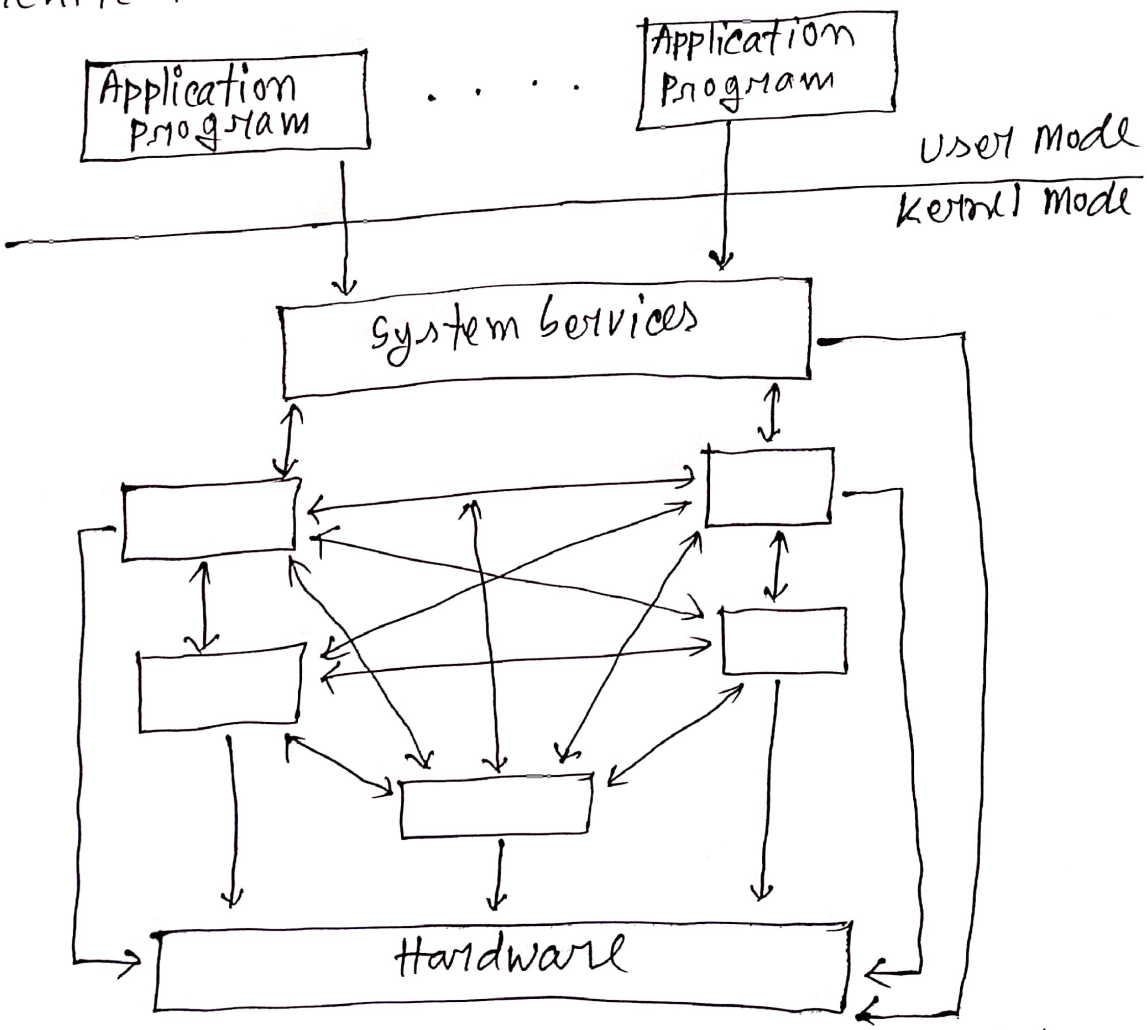
Ans. to the que. No: 6

(a)

In Microkernel architecture, small number of processes are allowed in the kernel. The device driver management, protocol stack, file system etc. may run in user space. Microkernel servers are called daemon process. In the kernel, memory protection, process scheduling. Inter process communication (IPC) purposes are served. In this architecture, the kernel grants some of the programs privilege to interact with parts of physical memory that are otherwise off limits to most programs. Microkernel architecture reduces the code size of the kernel and increases security and stability of the operating system.



In monolithic modular architecture, functions are loaded module by module. If any update required, the particular module is updated only. No need to update the whole OS. The main disadvantage of monolithic modular architecture is, the size of the kernel is huge and maintainability is poor. These problems are fixed in Microkernel architecture.



In modern days, Microkernel and monolithic modular architecture are used in combination. So we can choose combination of these two

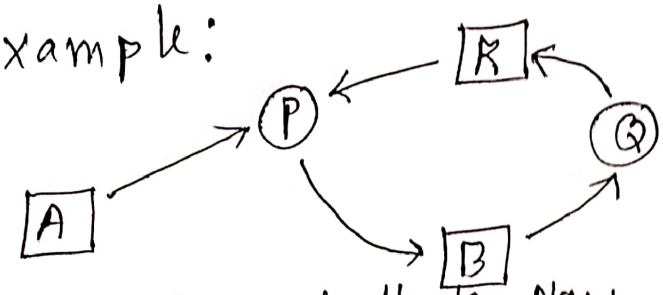
to design an operating system.

(b)

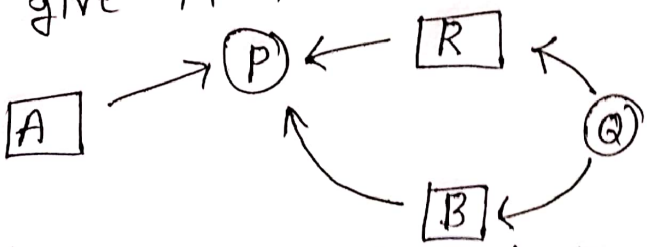
Following are the probable options for recovering from a deadlock:

i] Resourse Pre-emption: we can take back a resource from any process and give that resource to any other process. This is called resource pre-emption. we can recover from a deadlock by this.

For example:

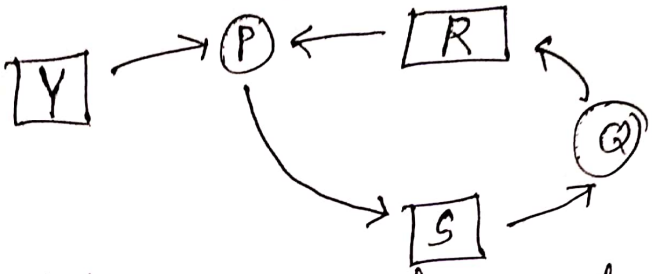


This is in deadlock. Now we can recover from the deadlock by taking back B from Q and give it to P.

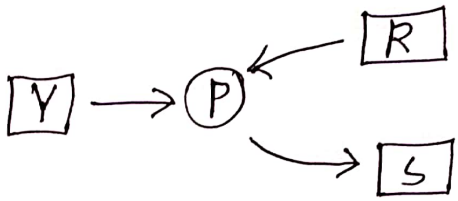


Deadlock is recovered. But the problem is, the Q's work will be lost. when Q will get the resource again, it should start its work from the very begining.

ii) Process killing: we can kill a process to recover from deadlock.



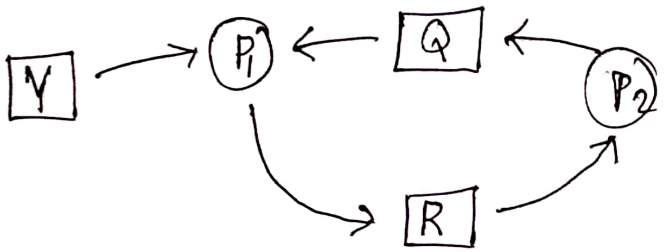
Now we will recover from deadlock by killing process Q.



But the same problem, loss of work, happen here. There is also other problem like taking decision about which process to kill. we can kill processes having -

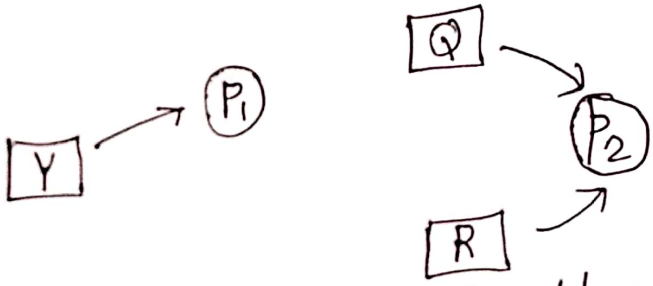
a) low priority
b) less connection
c) less time running
d) killing common node.

iii) Rollback: we can rollback one step or more than one step to recover from deadlock. In case of rollback, all work will not be lost. only the works of those steps rolled back will be lost. This is an advantage compared to other ways.



P1 : { Y
~~Q~~
Req: ~~R~~
}

2 steps rolled back.



So, now there is no deadlock.
Thus we can recover deadlocks.

(c)

Following are the steps of the equation that an OS uses to estimate CPU time for a process:

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

Here,
E_{n+1} = Estimated CPU time at n+1
t_n = Actual time taken by a process at n
E_n = Estimated CPU time at n
α = a constant between 0 and 1.
$0 \leq \alpha \leq 1$.

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

Here, E_0 = System average time

We have to choose the value of α very carefully. The main equation is,

$$E_{n+1} = \alpha t_n + (1-\alpha) E_n, \quad 0 \le \alpha \le 1$$

putting $\alpha = 0$,

$$E_{n+1} = E_n$$

So, no estimation is happening here.

putting $\alpha = 1$,

$$E_{n+1} = t_n$$

Actual time in used everywhere.

putting $\alpha = 0.5$,

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

So, we have to use $\alpha = 0.5$. It is used conventionally. The equation we have used is a recurrent equation.

Ans. to the que. No: 1

(a)

Kernel is the component of operating system that manages operations of processor and hardware. It is the primary interface between hardware and process. kernel is loaded first into the startup. It provides basic services for all other parts of the operating system. kernel is the essential part of an operating system. operating system can not perform its duty without kernel. So kernel is the must. That's why, operating system is called kernel.

Application program can't access hardware directly. It needs help of operating system to access hardware. operating system works as an interface between the application program and hardware components. An operating system uses system call to access hardware through operating system.

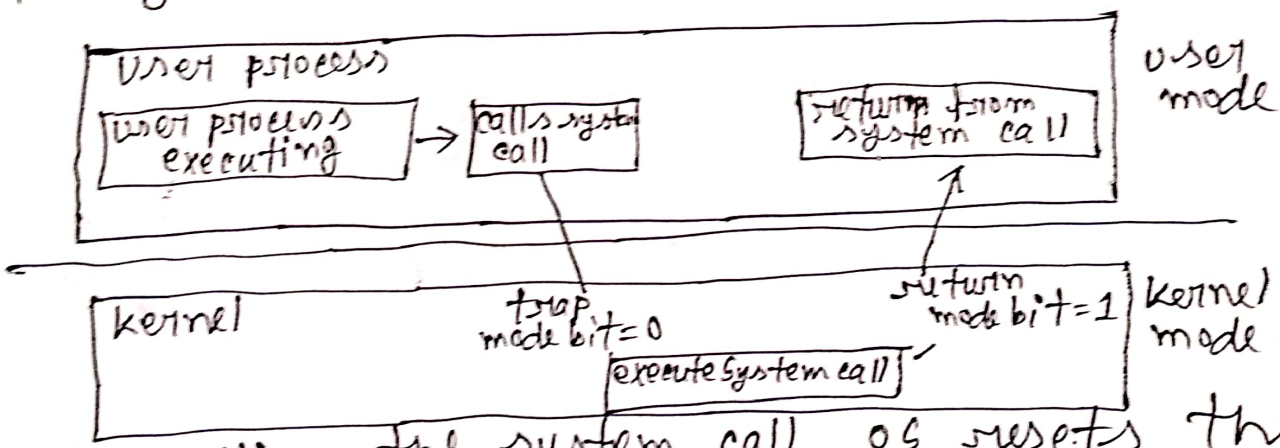
Student ID: 18.01.04.072
Course No: CSE3213 Rakib

A system call is the way in which an application program requests a hardware service from the kernel of the OS. When a program uses a system call, the OS switches into the kernel mode and executes that system call. OS finds the system call table which is saved into the kernel.

For example,

count = read(fd, butter, nbytes)

read() is a system call. This is used to read a file. fd is file descriptor, nbytes is the number of bytes in the file and butter is the location where read deposits the bytes.



After calling the system call, OS resets the trap bit into 0 and switches to the kernel mode. To protect the kernel from the user program ~~trap~~ mode bit is changed. OS executes the system call and return back to user mode by setting the mode bit to 1.

When we have called the read() system call, OS switched to kernel mode and look for the function in its system call library table, finds it and execute the call. After successful execution switch back to user mode.

(b)

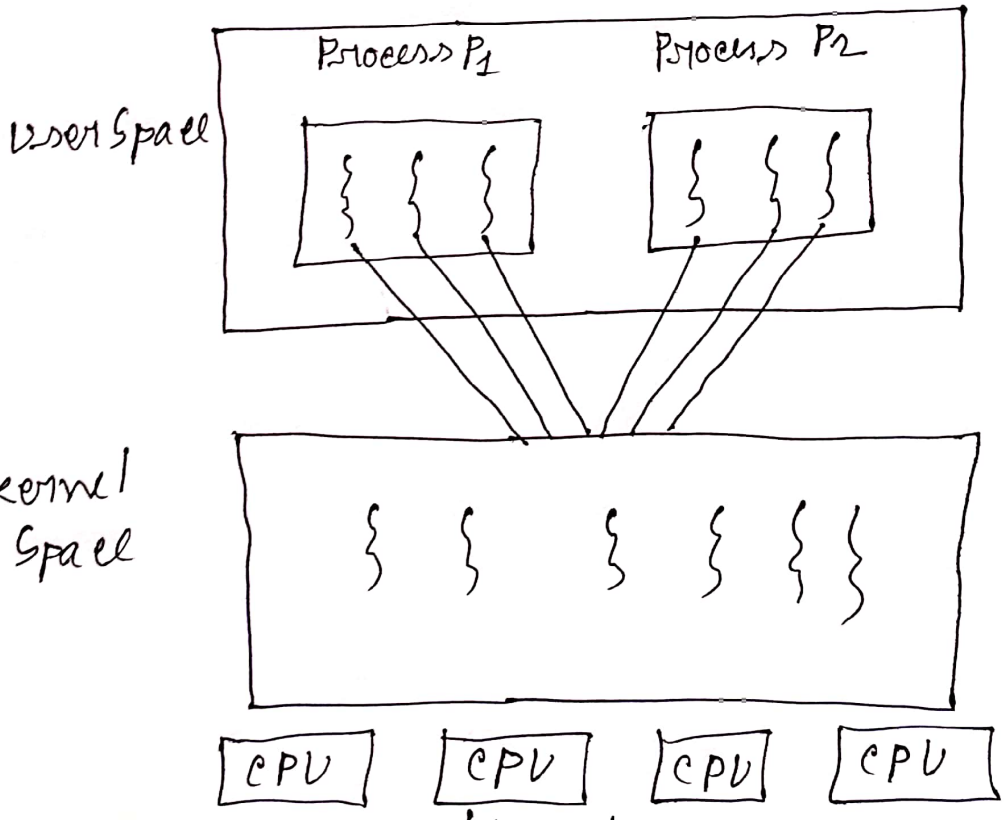
CPU utilization means using the resources of the computing system efficiently, don't let the system to sit idle. Operating system can ensure better cpu utilization by scheduling the processes properly. OS should schedule the processes in a way that the CPU can't sit idle. Average response time and average waiting time must be low and throughput must be high. OS should use proper scheduling algorithm to avoid race condition and deadlocks. Multiprocessor scheduling can be another way to use the cpu in a proper way.

For example, Process A and B are in the ready Queue. The CPU have to serve these processes in minimum time frame as possible. The user should have the feel like multi-tasking. The CPU can use

Round-Robin scheduling algorithm to serve these two process simultaneously divided into time quantums.

Kernel level thread architecture is performed by the kernel of the OS. It is directly supported by the OS. The kernel performs thread creation, scheduling and management in kernel space.

operating system introduce threads as CPU have only fixed cores.



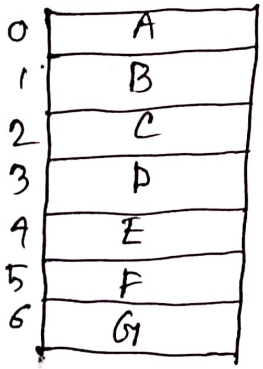
In kernel level thread, when a process is being executed, the kernel reads it the process supports multithreading. Then it

divides the process into multiple threads and loads it into the CPU. In kernel level thread architecture, processes are served by the threads in kernel space.

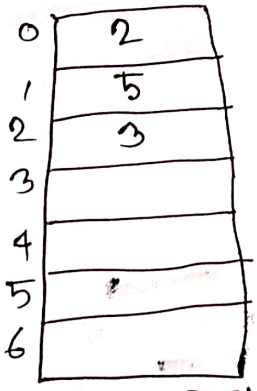
Ans. to the que. No: 3

(a)

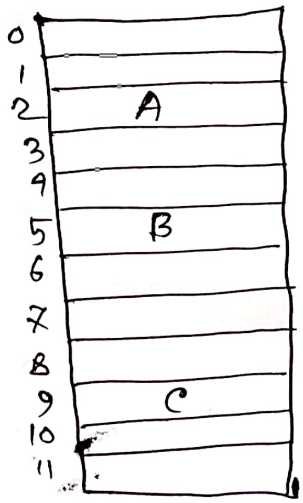
Page fault: Page fault occurs when CPU generate a logical address and where page no isn't present in the main memory. So for this particular page, the page table is empty. For example,



Logical memory



Page Table



Physical memory

If CPU generates a logical memory and request for page no 3 page fault occurs.

Given,

Page frame size = 128 KB

RAM Size = 1 GB
= (1 × 1024 × 1024) KB
= 1048576 KB

∴ No of frames = $\frac{1048576}{128}$
= 8192

Address space size of process = 256 MB
= (256 × 1024) KB
= 262144 KB

∴ No of pages = $\frac{262144}{128}$
= 2048

offset size = 128 KB

∴ No of bits for offset = 128 × 1024
= $2^7 \times 2^{10}$
= 2^{17}
= 17 bits

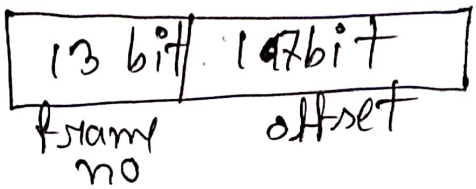
∴ No of bits for frame = 8192
= 2^{13}
= 13 bits.

∴ No of bits for pages = 2048
= 2^{11}
= 11 bits

So, virtual address format,

11 bit	17 bit
Page No	offset

Physical address format



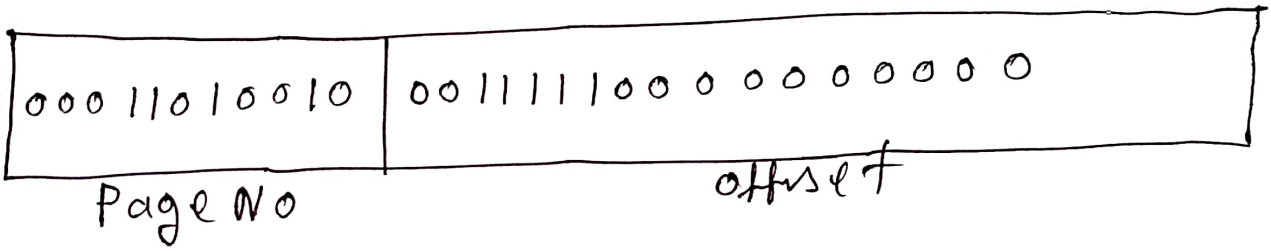
Frame is $318 = (000010011110)_2$

Page is $210 = (00011010010)_2$

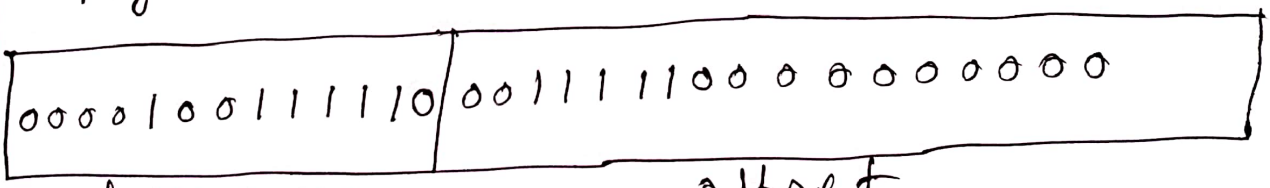
offset is $31 \times 2^{10} = 31744$

$= (00111110000000000)_2$

∴ Virtual address,



∴ Physical Address,

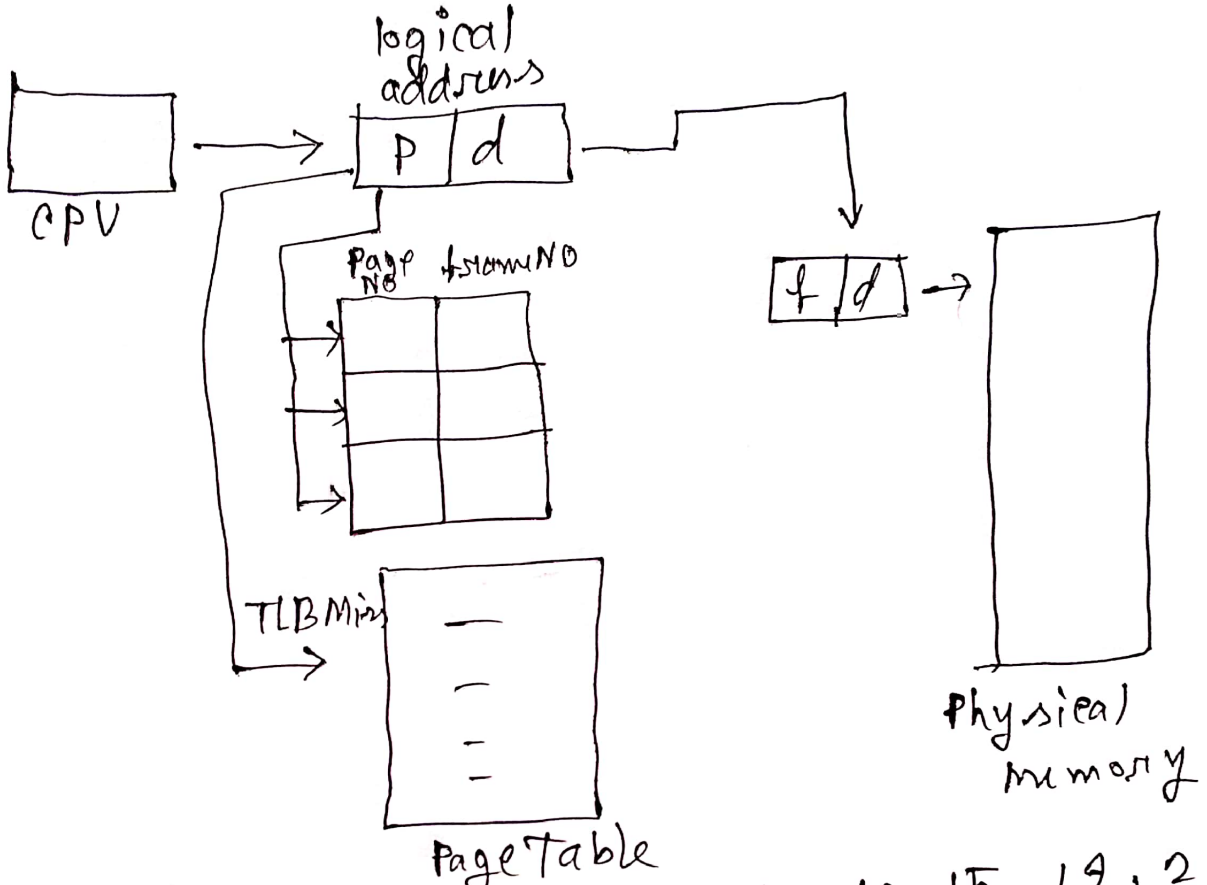


(b)

Translation Lookaside buffer:

when CPU request a particular page ~~it saves~~ frequently, CPU saves its page no and frame no in cache memory. In cache memory level1 and level2 has a table named TLB where these data are stored. In the entries of TLB

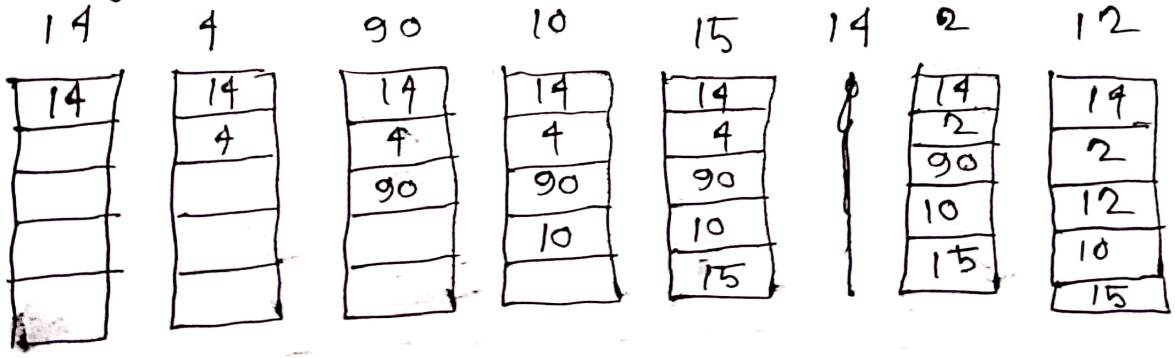
the bus lines are connected. So fetching frame no is needed less time.



Given Page requests; 14, 4, 90, 10, 15, 14, 2, 12, 0, 41, 7, 9, 7, 15, 0, 14, 7, 10, 5, 21, 3, 11, 5, 1, 41, 10, 90

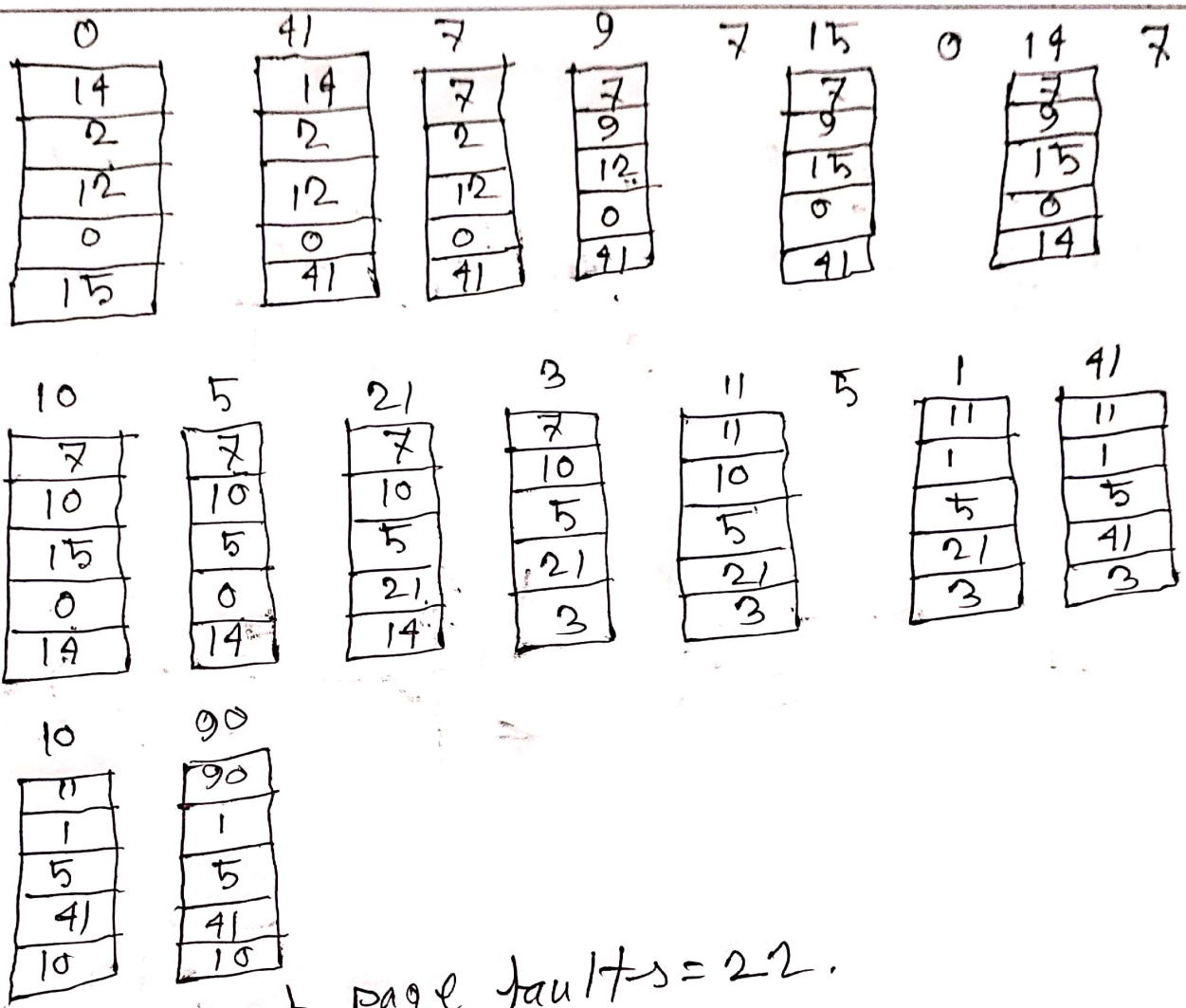
Page frame size is 5.

Using LRU,



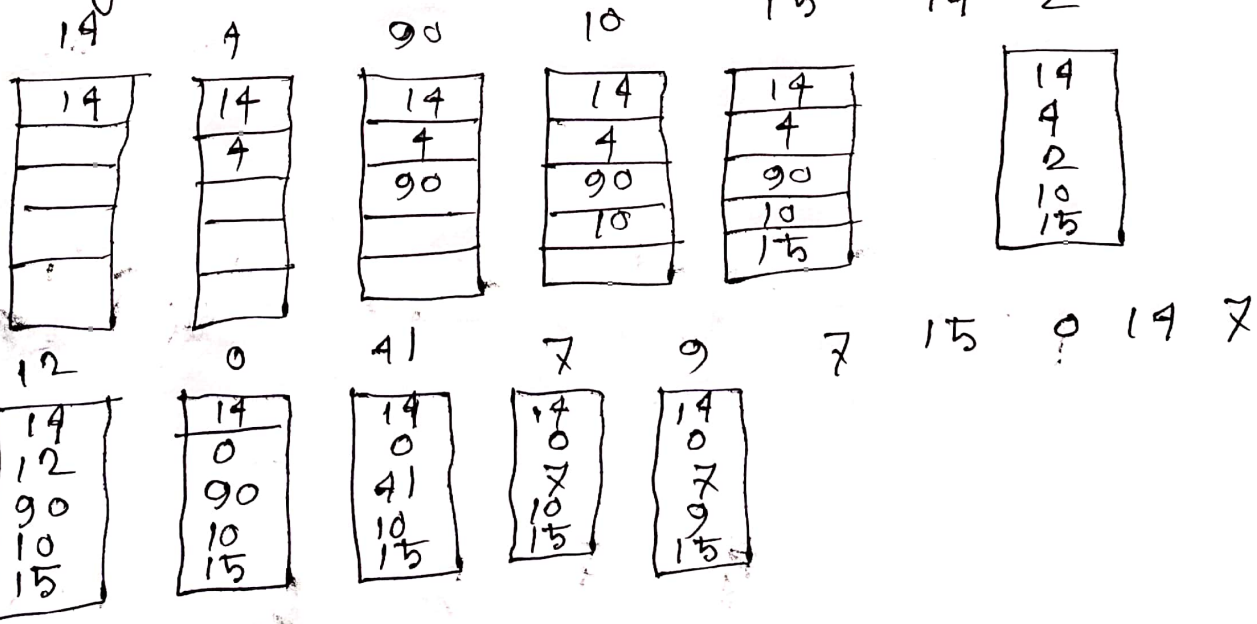
Student ID: 18.01.04.072

Course No: CSE3213 Rakib



So no. of page faults = 22.

using optimal.



10	15	21	11	5	1	41	10	90
10	10	10	10	10	10	10	10	10
0	5	5	5	1	1	1	1	1
7	7	21	21	21	21	41	41	41
9	9	9	3	3	3	3	90	90
15	15	15	11	11	11	11	11	11

No. of Page fault = 19.
For this particular example optimal page replacement works better. But optimal page replacement isn't implementable in real life as we have to know the future requests.

Ans. to the que. No: 5

(b)

Given requests.
2 16 3 9 80 70 37 32 416 10
210 19 290 15 33
current head position: 111

FCFS:
Total movement = (111 - 2) + (16 - 2) + (16 - 3) + (9 - 3)
+ (80 - 9) + (70 - 80) + (70 - 37) + (32 - 37)
+ (32 - 416) + (10 - 196) + (10 - 210) + (19 - 210)
+ (19 - 290) + (15 - 290) + (~~zero~~ - (15 - 33)
=

Student ID: 18.01.04.072

Course No: CSE3213 Rakib

Ans to the qu. NO: 2

(a)

```
#define N 5
#define left (i+N-1)%N
#define Right (i+1)%N
#define Thinking 0
#define Hungry 1
#define Eating 2
int state[N];
int Lock=0;
semaphore S[N];
void philosopher(int i){
    while(True){
        think();
        take_forks(i);
        eat();
        put_forks(i);
        eat();
    }
void take_forks(int i){
    while [Lock==0)
    {   Lock=1;
        state[i] = Hungry;
        test(i);
        Lock=0;
        down(S[i]); }
```

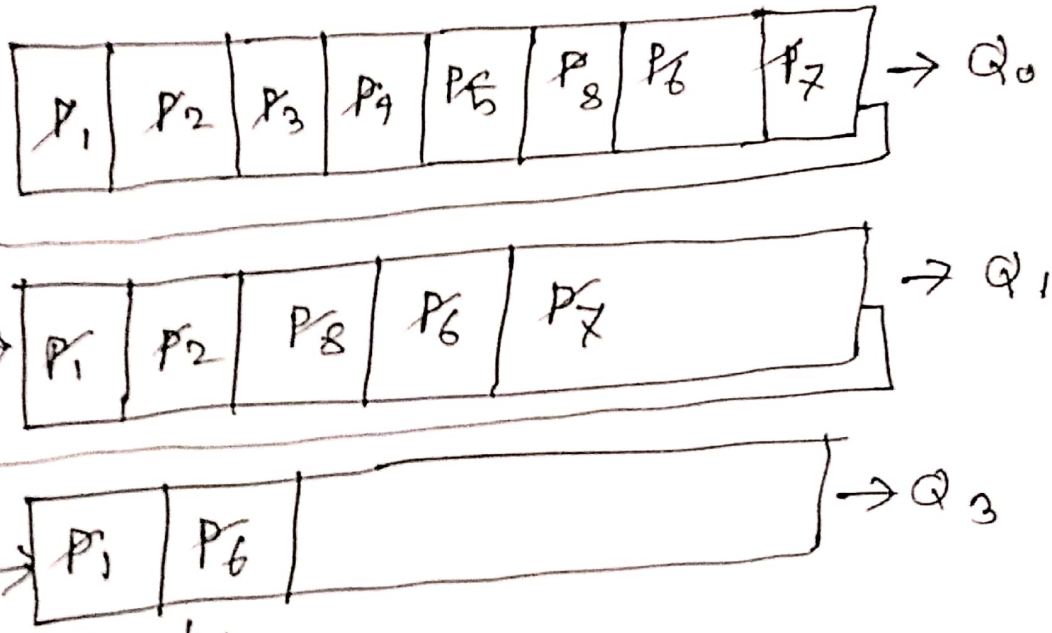
```
void put_forks (int i){
    while(Lock==0)
    {
        Lock=1;
        state[i]=Thinking,
        test(left);
        test(Right);
        Lock=0;}
    }
void test(int i){
    if(state[i]==Hungry && state[left]
        !=Eating && state[Right]!=
            Eating)
    {
        state[i]=Eating;
        up(S[i]);}
    }
```

My algorithm is highly effected. I am tracking ~~the~~ each philosopher state using S[i] semaphore. As per the problem if anyone is in eating state then the left and righ person can't be in eating state. I checked this using if condition. So when philosopher a is eating, I can't change his state in eating.

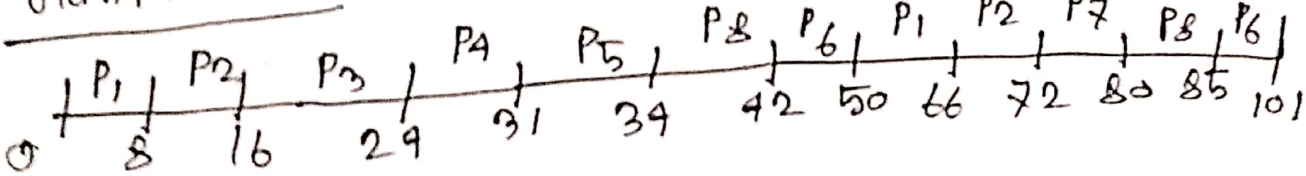
(b)

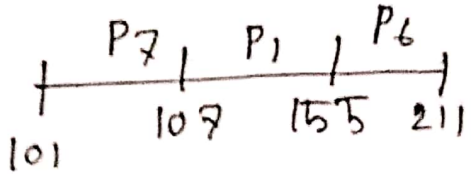
Let there are 3 queues.

1. Q_0 = RR with time quantum 8
2. Q_1 = RR - - - - - - 16
3. Q_3 = FCFS



Gantt chart:





Student ID: 18.01.04.072

Course No: CSE 3213 Rakib

Process	A.T	CPU T.	W.T.
P_1	0	72	0+42+4 = 83
P_2	6	14	2+50 = 52
P_3	9	8	7 = 7
P_4	12	7	12
P_5	20	3	11
P_6	28	80	14+35+54 = 103
P_7	68	19	4+21 = 25
P_8	25	13	9+38 = 47

∴ Average waiting time of the processes are

= (83+52+7+12+11+103+25+77)/8

= 42.5.

So from average waiting time we can see the efficiency of our design is not that good. Because the average waiting time is high.