Department	:	CSE
Program	:	B. Sc. in CSE
Course No.	:	CSE-3213
Course Title	:	Operating System
Examination	:	Semester Final
Semester (Session)	:	Fall-2020
Student No.	:	18.01.04.129

Signature and Date:

Md. Anwarul Habib

24/10/2021

18.01.04.129
CSE-3213
Md. Anwarul Habib

Ans. to the Ques. No. 3

(a) A page fault occurs when a program requests an address on a page that is not in the current set of memory resident pages.
What happens when a page fault occurs is that the thread that experienced the page fault is put into a wait state while the operating system finds the specific page on disk and restores it to physical ~~address~~ memory.
When a thread attempts to reference a non-resident memory page, a hardware interrupt occurs that halts the executing program. The instruction that referenced the page fails and generates an addressing exception that generates an interrupt.

The ISR that gains control at this point and determines that the address is valid, but that the page is not resident. The OS then locates a copy of the desired page on the page file and copies the page from disk into a free page in RAM. After a successful copy, OS allows the program thread to continue on.

Given that,

RAM Size: 1 GB $\rightarrow 1 \times 2^{30}$

Process Size: 256 MB $\rightarrow 2^8 \times 2^{20} \rightarrow 2^{28}$

Page frame Size: 128 KB $\rightarrow 2^7 \times 2^{10} \rightarrow 2^{17}$

No. of Page frames in RAM: $\frac{2^{30}}{2^{17}} = 2^{13}$

" " " " in Process: $\frac{2^{28}}{2^{17}} = 2^{11}$

∴ Total RAM frames $= 2^{13} = 8192$

∴ Total Process pages $= 2^{11} = 2048$

∴ Logical Address:

Page No.	Offset

here, Page No. = 210 that will be presented by 11 binary bits, $(210)_{10} = (00011010010)_2$

Offset $= 31 \times 2^{10} = 31744$ which will be presented by 15 binary bits, $(31744)_2 = 001111100000000$

$(210)_2$	$(31744)_2$
00011010010	001111100000000

∴ (Logical Address).

Md. Anwarul Habib

Physical Address :

Frame No.	Offset

here, Frame No. = 318 that will be represented by 13 binary digits.

$(318)_2 = 0000010011110$

Offset = $31 \times 2^{10} = 31744$, which will be represented by 17 binary bits.

$(31744)_2 = 00111110000000000$

$(318)_2$ $(31744)_2$

0000100111110	00111110000000000

∴ Physical Address.

(Ans).

(b). A translation lookaside buffer (TLB) is a memory mapping cache that is used to reduce the time taken to access a user memory location. It is a part of the processors Memory Management Unit (MMU). The TLB stores the recent translations of virtual memory to physical memory. A TLB is organized as a fully associative cache and typically holds 16 to 512 entries. Each TLB entry holds a virtual page number and its corresponding physical page number. The TLB is accessed using the virtual page number. If the the TLB hits, it returns the

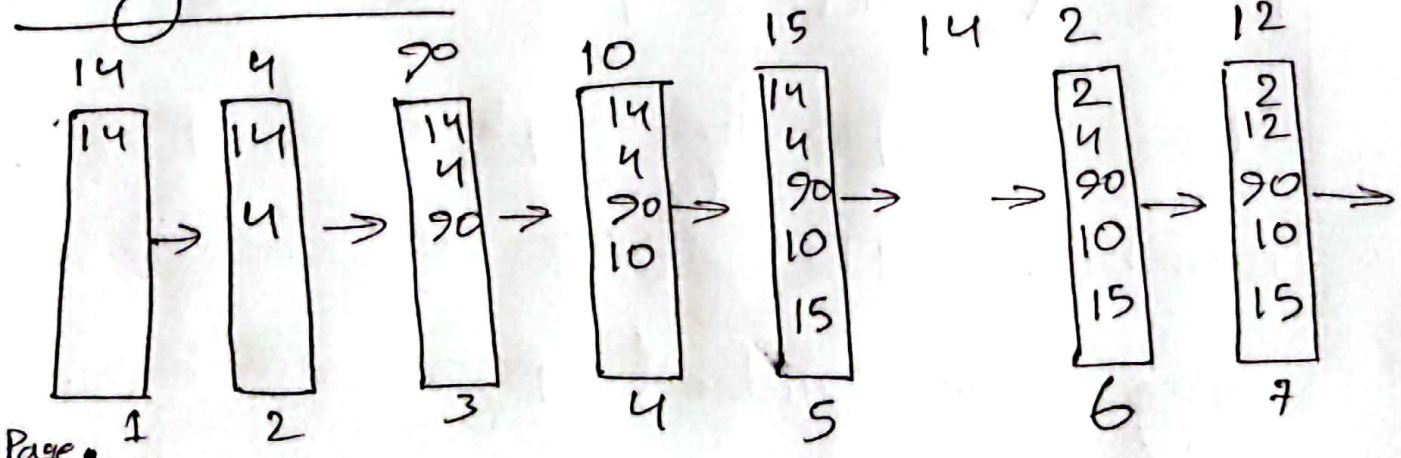
physical page number. Otherwise the processor must read the page table in physical page number.

Given that,

Page Frame size : 5

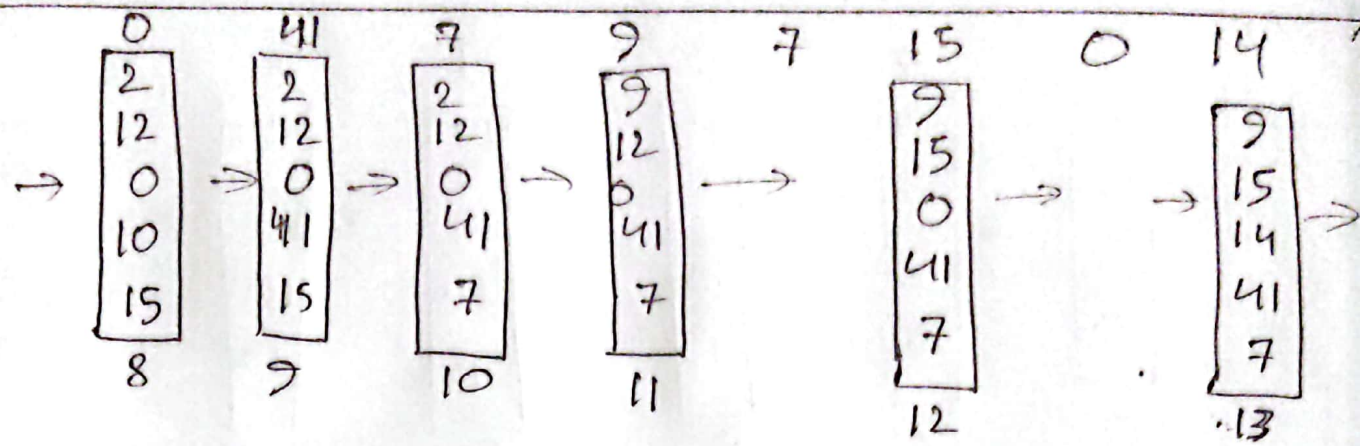
Page Requests : 14, 4, 90, 10, 15, 14, 2, 12, 0, 41, 7, 9, 7, 15, 0, 14, 7, 10, 5, 21, 3, 11, 5, 1, 41, 10, 90.

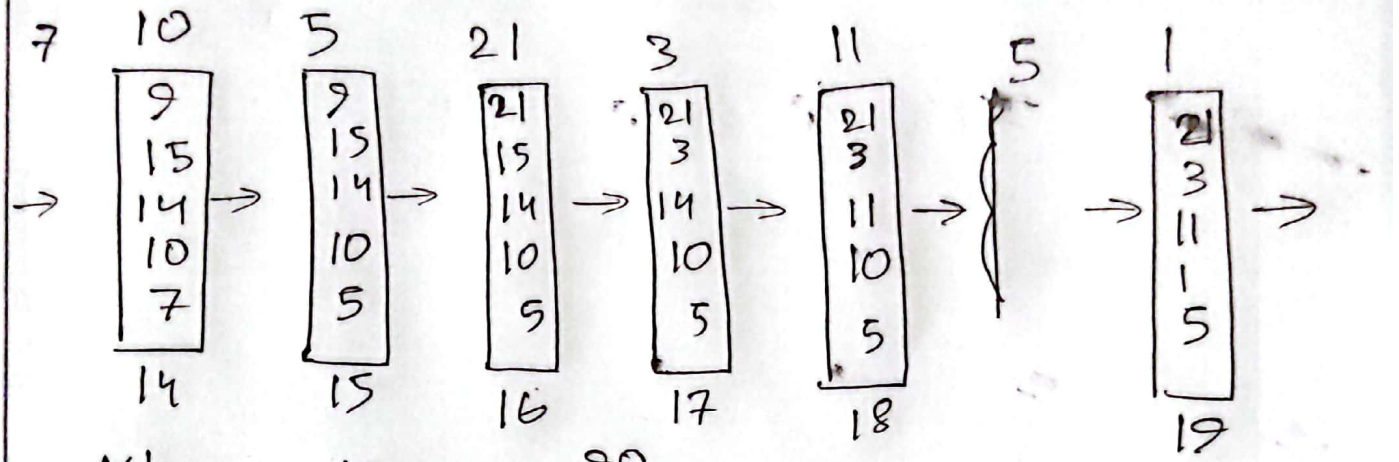
Using LRU :-

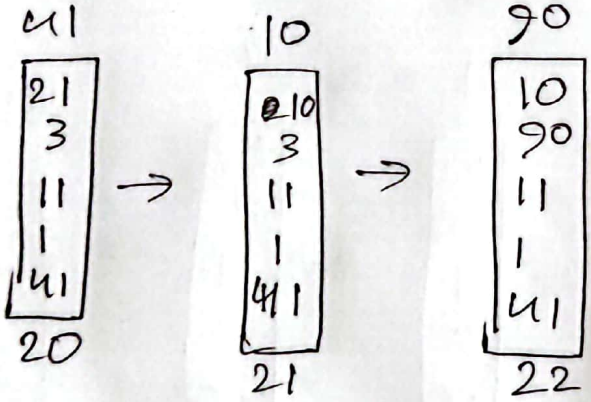


Page Faults:

18.01.04.129
CSE-3213
Md. Anwarul Habib

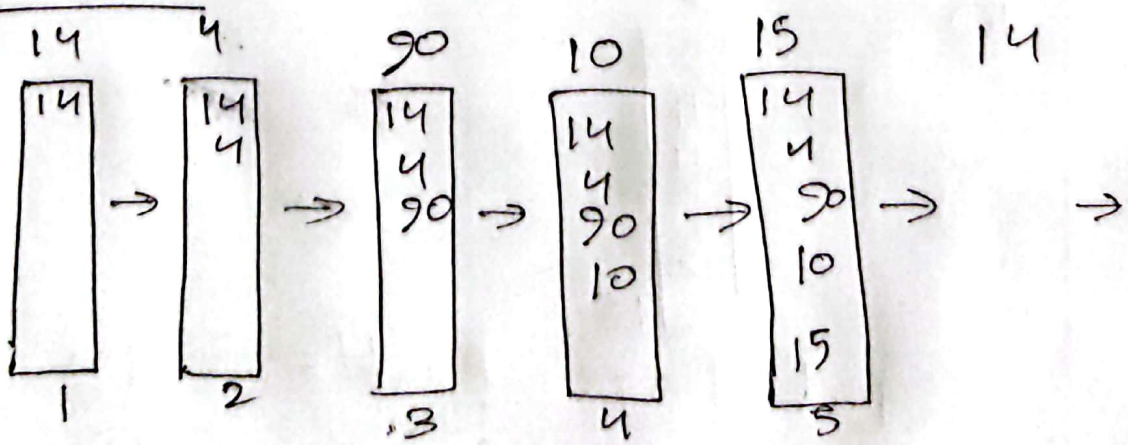






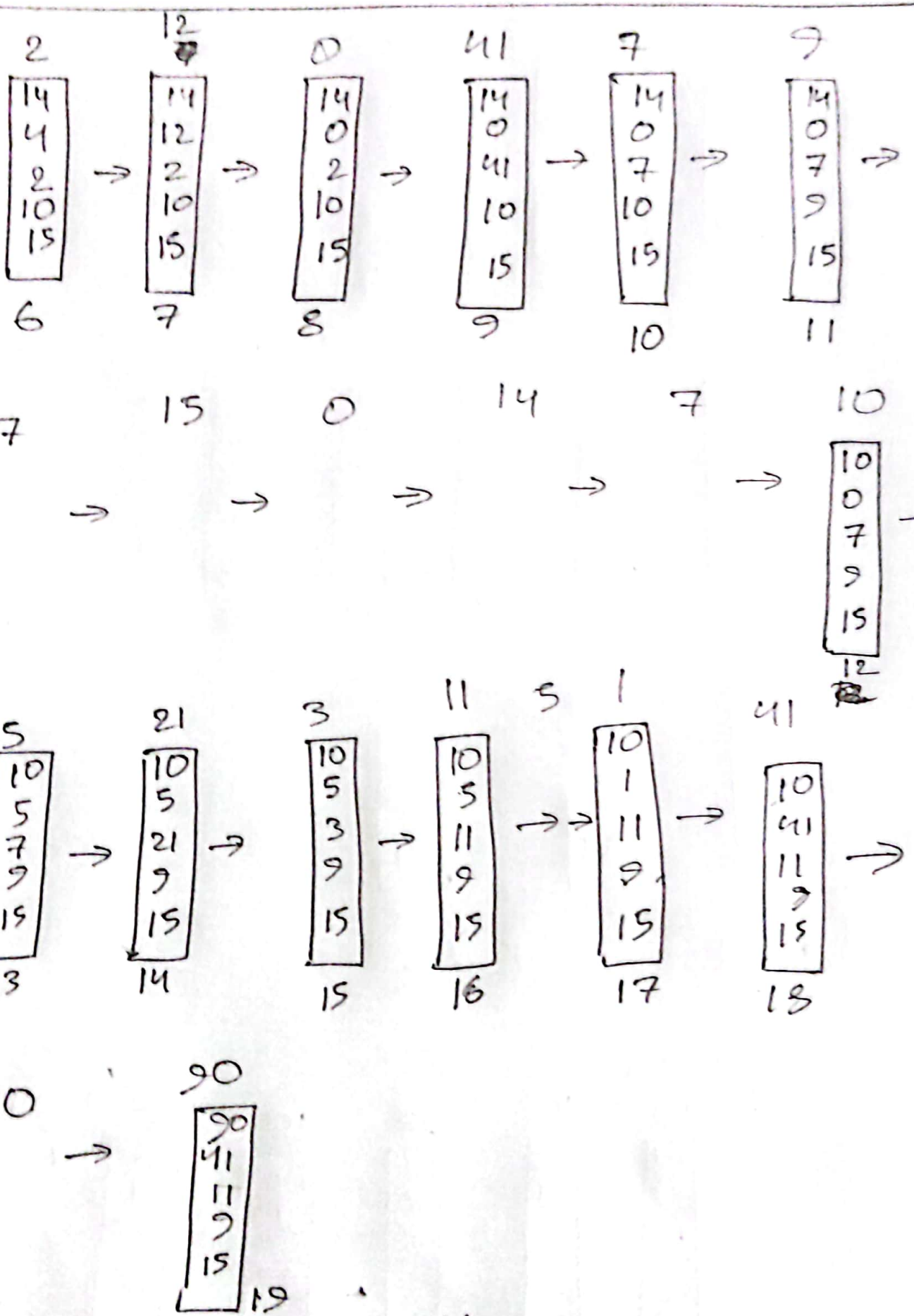
Total Page faults = 22

Using Op:



18.01.04.129
CSE-3213

Md. Anwarul Ibbi



∴ Total Page Faults = 19

So, OP has less page faults than LRU.

(Ans). 108

Md. Arwarul Habib

Ans. to the Ques. No. 6

(a) Microkernel and Monolithic modular architecture comparison.

Microkernel	Monolithic
Some important modules are stored into kernel.	Full OS program is stored in a single file
IPC, Scheduler, Memory handlers etc are put into kernel. Rest of those are in user mode.	All the modules are put into kernel and the management also done by kernel
Updates are easier. Only a small portion is updated.	Updates are horrible since the entire system needs to be updated.
Relatively slower	Faster processing

Error finding and fixing is easier.	Error troubleshooting takes much effort.

While designing an OS, I will choose the Microkernel architecture. Because the development and maintenance is easier in this system than the Monolithic architecture. Also making important updates will be much more easier on a microkernel system. Also there is much control over user mode on how user applications access hardware resources.

(b) It is relatively easy to prevent deadlock than recovering the system when it occurs. When deadlock happens, there are not much options on our hand. But we can take the following steps to recover the system from deadlock.

(i) Resource Preemption :- we can preempt the wage of deadlock participating resources. For example, we can define a time quantum which is the maximum time length a single process can hold a particular process. When the time quantum ends, we can give chance

to another process. Also process preemption ~~of the~~ may be done based on priority.

(ii) Process killing : Processes can be killed in border to break the deadlock cycle. Processes with lower priority or a process with least amount of run time can be killed.

(iii) Rolling Back :- Processes can be rolled back to some certain step until the system becomes safe. Process log book can be used to identify resource requests in each step and thus

rolling back can be done.

(c) CPU brust time cannot be predicted but it can be estimated based on usage, by the following equation:

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

Where,

→ E_{n+1} is the estimated CPU Time at $n+1$.

→ t_n is the actual time taken by a procen at n.

→ α is a constant, $0 \leq \alpha \leq 1$

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

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

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

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

$= \alpha t_n + (1-\alpha)\alpha t_{n-1} + (1-\alpha)^2 \alpha t_{n-2} + (1-\alpha)^3 E_{n-2}$

$\vdots$

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

[E_0 = system Average time]

If, $\alpha = 0$ put on (1),

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

$\therefore E_{n+1} = E_n = E_{n-1} = \ldots = E_0$

So, $\alpha = 0$ leads to no estimation.

If, $\alpha = 1$ put on (1),

$E_{n+1} = t_n$; which is equal to previous estimation time.

For α, ideal value is 0.5.

If $\alpha = 0.5$ put on (1),

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

which takes 50% from the past estimation and 50% from the actual taken time on the previous step.

18.01.04.129
CSE-3213
Md. Anwarul Habib

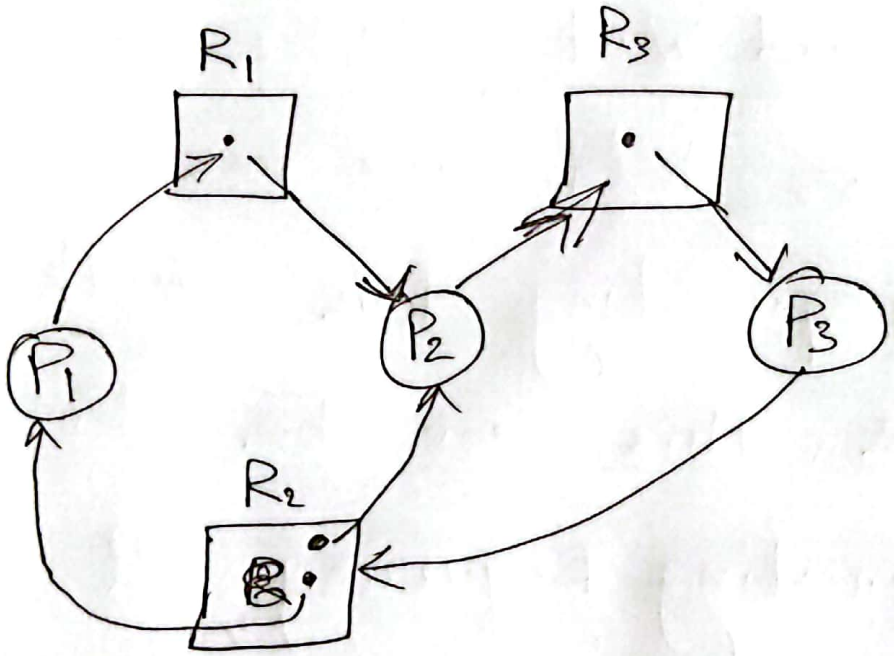
Ans. to the Ques. No. 4

(a) When a set of process creates a circular dependency, with the requested and holded resources, they create a deadlock.

We know that, process can hold resources when it is currently being used. It can also request some instance of resources when it needs. Requested resources may not be available at that time. Thus it waits until the requested resource is free. A Resource Allocation Graph (RAG) can be drawn from those informations. If there appears a cycle in RAG, it is a

16
15

deadlock. In a deadlock, a process requests some resource acquired by another process on the same set. Following is a RAG with deadlock.



Here, R_3, P_3, R_2, P_2 creats a circular dependency.

Hence, $D_1 = \{R_3, P_3, R_2, P_2\}$ is are in a deadlock.

Deadlocks can be prevented by the following steps :

(i) Mutual Exclusion: Mutual exclusion must hold for non-sharable resources.

(ii) Hold and Wait : Make sure that no process holds any resource while requesting for another resource. This can be done by—

⇒ ~~Requesting~~ Requiring all the resources before it starts execution

⇒ Allowing to request the resource only when they are not allocated by any other resource.

(iii) Preempt ~~R~~ resources when they create a circular dependency

and before that release all the resources currently allocated by the cycle.

~~(iv) &r~~

(b) (i) Asymetric Multiprocessor Scheduling :

In this system, there is one dedicated CPU for scheduling other processes, it is called the master server. But there is a risk, if the master server fails somehow, all other processors become idle.

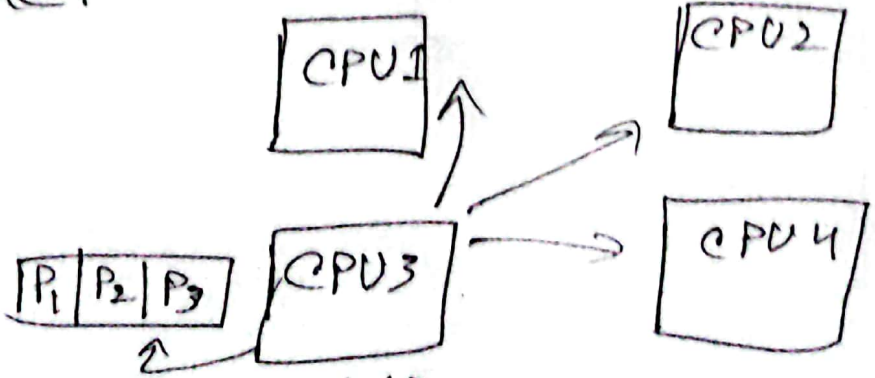


Fig: AMP.

(i.i) Symetric Multiprocessor Scheduling:

In this system, there is no dedicated CPU for scheduling, but each processor has their own scheduler and a common process queue.

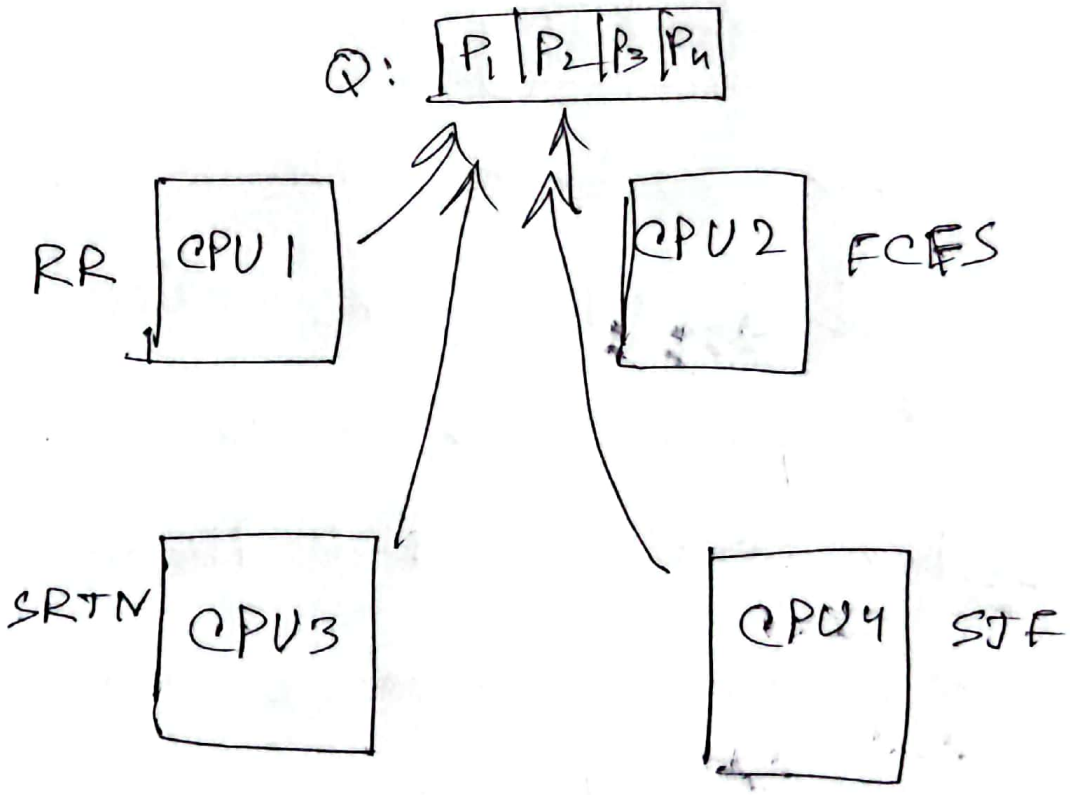


Fig: SMP

(iii) Push Migration : In this system there is a dedicated software that checks each processor queue after a certain amount of time and if it finds any imbalances, it balances the load.

Pull Migration :- No dedicated software and when a processor becomes free/idle, it itself checks the CPU queues and access them to load balancing. It loads the queue first before checking other CPU queues.

Ans. to the Ques. 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_fork(i);
        eat();
```

```
        put_forks (i);
    }
}

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]);
    }
}
```