Assessment Fall 20-21

Syeda Nowshin Ibnat

ID: 17183103020

Intake: 39

Section: 1

Course Code: CSE 309

Course Title: Operating Systems

Semester: Fall 20-21

1(a) question Answer

Given.

```
Process P0 {
wait (lock2)
Signal (lock1);
wait (lock2);
Signal (lock2);
}
```

```
Process P1 {
wait (lock1);
wait (lock2);
Signal (lock2);
signal (lock1);
}
```

Here, Both process will be locked forever.

Process P_0 and Process P_1 both.

From P_0 : without lock(2) acquire in P_2 will wait for lock 2. P_2 is giving signal of lock2 in the last so it will be acquire at last.

1(b) question Answer

Part-1

To overcome some of the difficulties with fixed partitioning, an approach known as dynamic partitioning was developed. When a process is brought into main memory, it is allocated exactly as much memory as it requires and no more for dynamic partitioning. Eventually it leads to a situation in which there are a lot of small holes in the memory. As time goes on, memory becomes more and more fragmented, and memory utilization declines. This problem is referred to as "external fragmentation", indicating that the memory that is external to all partitions becomes increasingly fragmented.

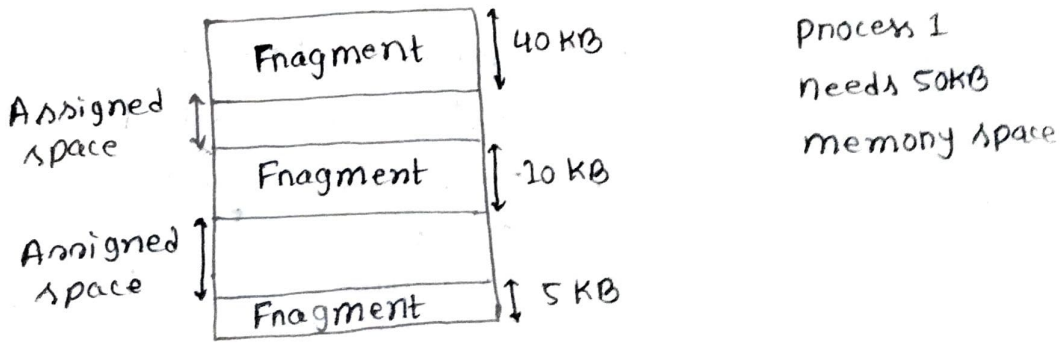


Fig: External Fragmentation

We can see from the figure that there is sufficient memory space (55 KB) to execute a process-1 (50KB mandated), but the ~~strioe~~ ~~stong~~ storage (fragment) is not adjacent.

Part-2

Now, to overcome external fragmentation, we can use "paging". Paging is a memony management scheme that avoids external fragmentation and ~~need for~~ the need for compaction. Whereas segmentation does not avoid it. concept of paging comes from the concept of segmentation. Paging solves the considerable problem of fitting memony chunks of varying sizes onto the backing stone. There is no possibility of external fragmentation on paging because partitioning ~~is not happ~~ doesn't happen here. In paging page size and frame size are same. So, the page we are using to stone are utilizing the frame 100%. That's why there is no need of partitioning. ~~So, Hence,~~ So, no chance of external fragmentation.

Suppose, we have 10 instructions and page size is 4.
The we will need:

Page No	instructions
Page 0	4
Page 1	4
Page 2	2
Page 3	

} no need of partitioning

Here, for page 3 there will be an internal fragmentation.

But there will be no ~~exe~~ external fragmentation hence ~~we~~ there is no need of partitioning.

2(a) question Answer

Part-1

Given, 3 processes = P_1, P_2, P_3

4 types of resources = R_1 (has 1 instance)

R_2 (has 2 instances)

R_3 (has 1 instance)

R_4 (has 3 instances)

Resource Allocation Graph

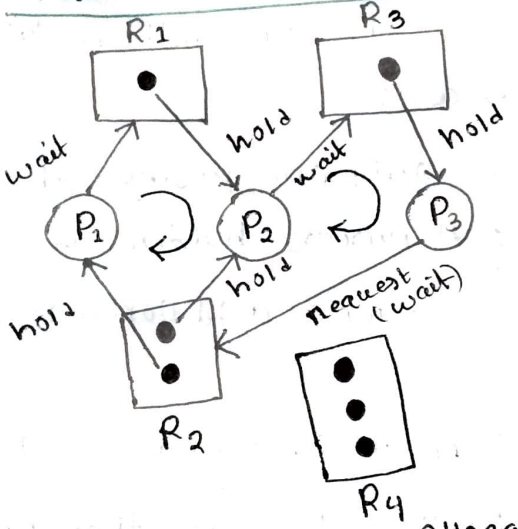


Figure: Resource-Allocation graph

From the figure above we can see, two minimal cycles exist in the system:

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

Processes P_1, P_2 and P_3 deadlocked. So, we can say that deadlock has occured in the system.

Part-2

Explanation: From the figure we can ~~see~~ see process P_2 is waiting for the resource R_3, which held by process P_3. ~~Othe~~ & on the other hand, process P_3 is waiting for either Process P_1 or process P_2 to release resource R_2. In addition, Proces P_1 is waiting for process P_2 to release resource R_1. So, from this scenario we can say deadlock ~~or~~ occured here. Deadlock can arise if 4 conditions hold simultaneously. This 4 conditions are:

(1) **Mutual exclusion:** only one process at a time can use a resource.

(2) **Hold and wait:** A process holding at least one resource is waiting ~~for~~ to acquire additional resources hold by other processes.

(3) **NO preemption:** A resource can be release only voluntarily by the process holding it, after that process has completed its task.

(4) **Circular wait:** When there exists a set $\{P_0, P_1, P_2 \ldots\}$ of waiting processes.

2(b) question Answer

Given,

logical address = 5 ~~as~~ pages

~~Page~~ page size = 4 bytes

Physical memony contains = 8 frames = 32 bytes

Use pnognam consists ~~of~~ = 20 instructions
(a, b, c, ... n, s, t)

~~Eac~~ Each instructions = 1 byte

Free frames are = 1, 2, 3, 5, 6

(i)

There are 5 pages in the page table. Since, there are 5 pages (Page 0 - Page 4).

(ii)

Page size = 4 bytes = 32 bits = 2^5 bits

There are total 8 frames. So we need 3 bits to request 8 frames. There is also a valid/invalid bit.

So, total = 3+1 = 4 bits in each table entny.

(iii)

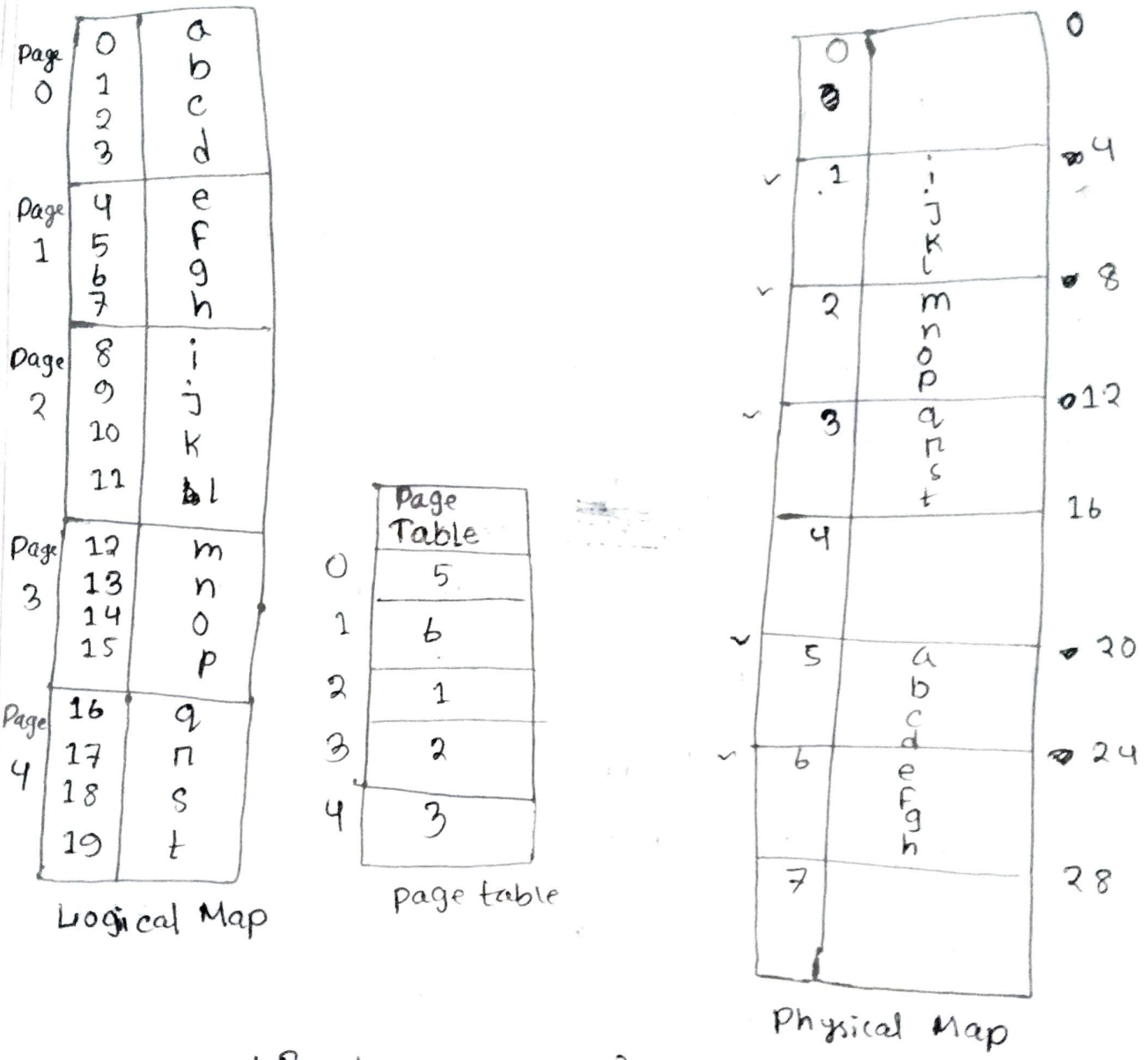


Figure: Logical and physical maps and page table

(iv)

Physical Address for the instructions: a, c, F, 0

	Page number	Page offset
a →	0	0
c →	0	2
F →	1	1
0 →	3	2

Physical address: a = 5×4+0 = 20 → Fname number

c = 5×4+2 = 22

f = 6×4+1 = 25

0 = 2×4+2 = 10

Here,
Physical Address = Fame number × Page size + offset

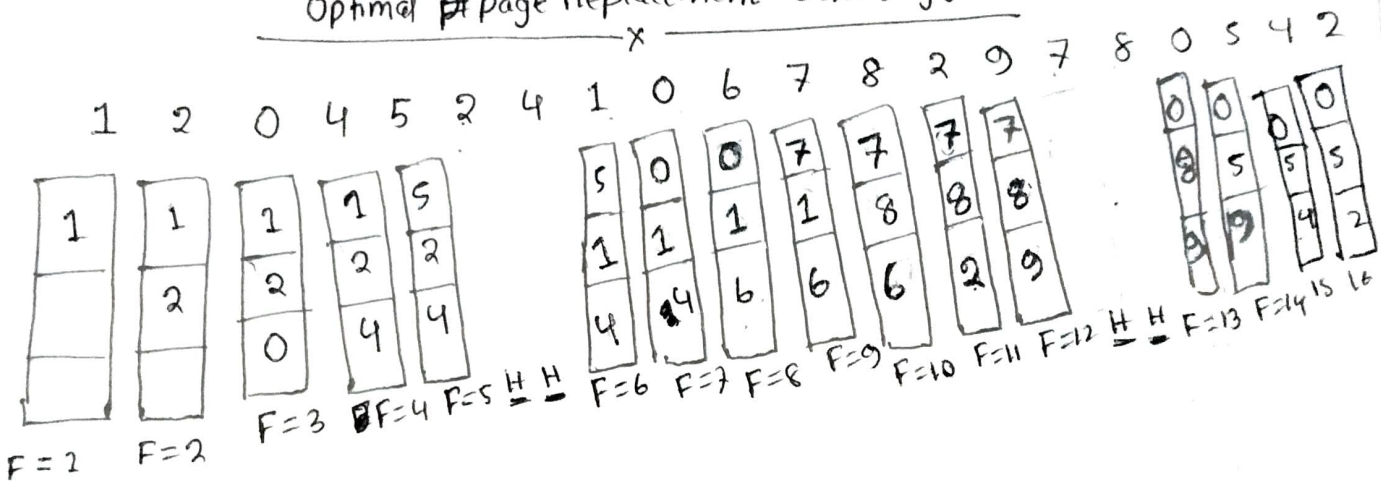
2(c) question Answer

Given, page fname = 3

Here, X = 0 My ID = 20 (last)
Y = 2 (second last)

Reference string: 1 2 0 4 5 2 4 1 0 6 7 8 2 9 7 8 0 5 4 2

Optimal page replacement Strategy



(i)

There are 4 page Hits.

(ii)

There are 16 page faults.

2(d) question Answer

Part-1

Given,

Disk drive has = 5000 cylinders (0 - 4999)

Serving a request at cylinder = 143

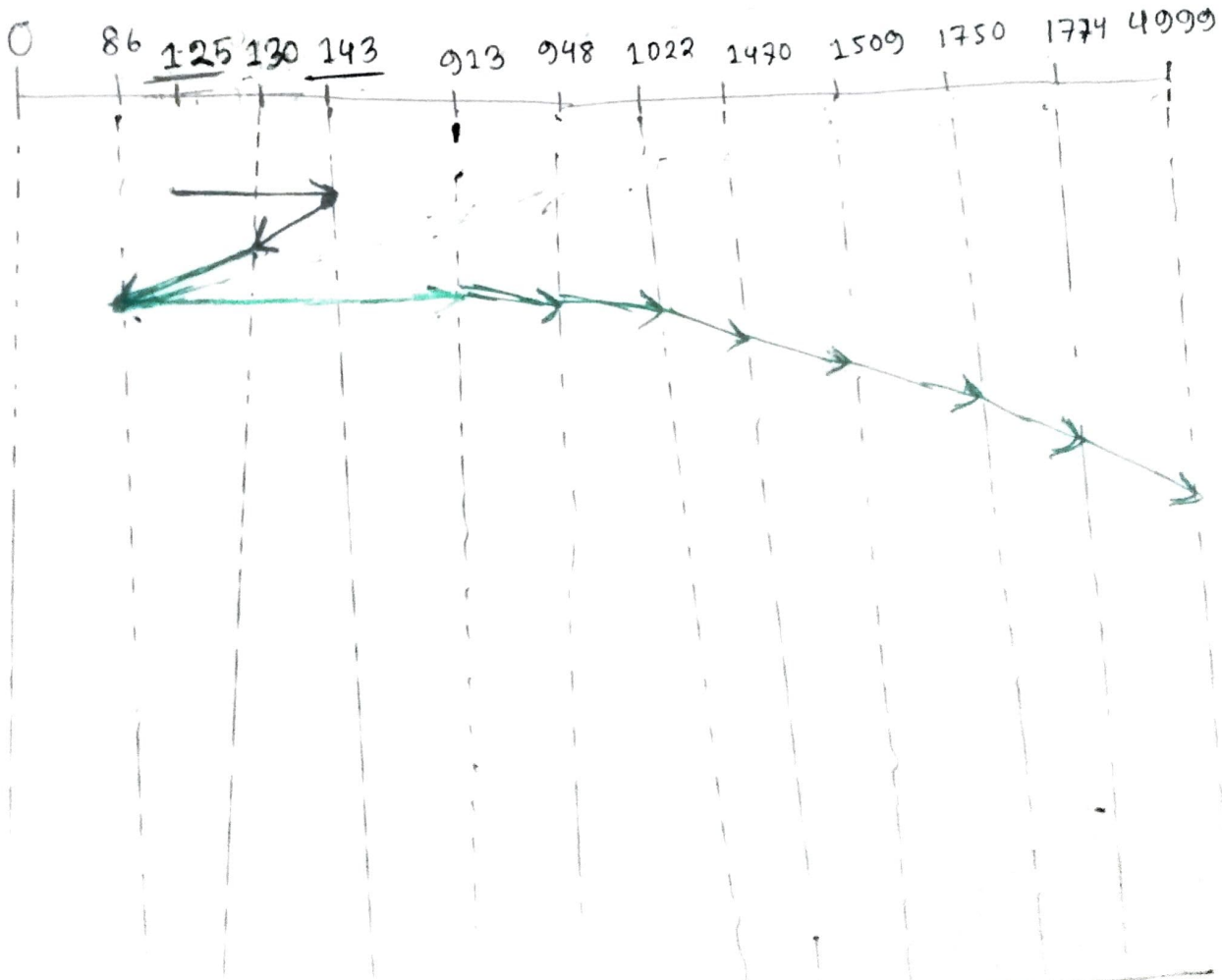
previous request = 125

Queue of pending request in FIFO order is:

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

(i)

SSTF (Shortest Seek time First)



Here, The SSTF Schedule is = ~~143, 86, 1470.~~

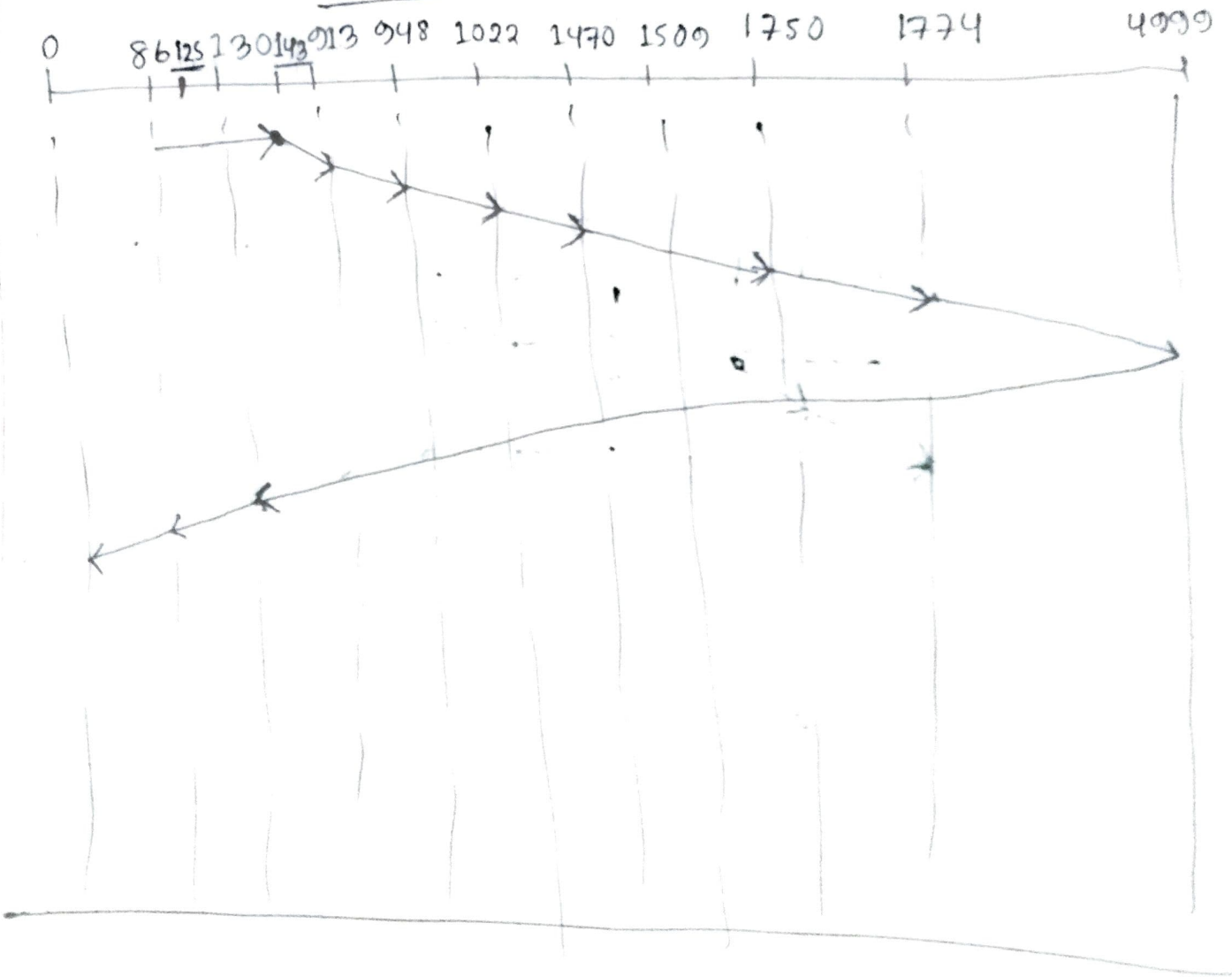
143, 130, 86, 913, 948, 1022, 1470, 1509, 1750, 1774.

Movements = 13 + 44 + 827 + 35 + 74 + 448 + 39 + 241 + 24

∴ Total distance = 1795

(ii)

SCAN



Here, SCAN Schedule is =

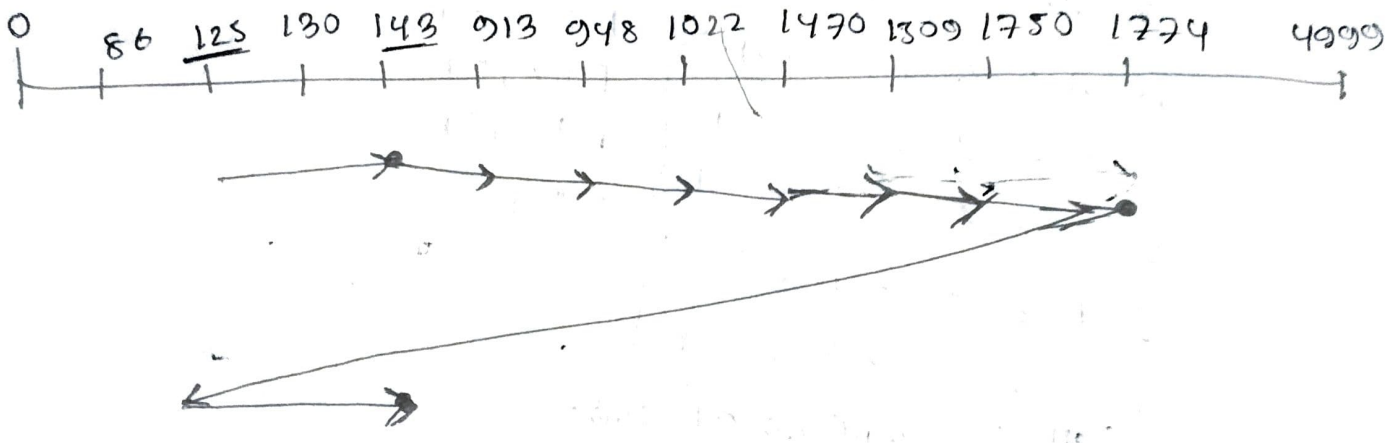
143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 130, 86

Movements = 770 + 35 + 74 + 448 + 39 + 241 + 24 + 3225 + 4869 + 44

Total = 9769

(iii)

C-LOOK



C-LOOK Schedule:

143, 913, 948, 1022, 1470, 1509, 1750, 1774, 86, 130

Movements = 770 + 35 + 74 + 448 + 39 + 241 + 24 + 1688 + 44

Total distance = 3363.

Part -2

We can't tell actually which algorithm is the most effecient one. Because, each algorithm performs good in different scenario. So we can't tell that one of them is efficient. SSTF is common and has a natural appeal. SCAN and C-SCAN perform better for systems that place a heavy load on the disk. This two provide less starvation.

So, there are different task performs by these different disk scheduling algorithm.

- FCFS every request get fair chance.
- SSTF decrease average response time.
- SCAN provide high throughput.
- CSCAN provides more uniform wait time compared to SCAN.
- So, we can't tell which one is the effecient among them.

3(a) question Answer

TLB or Translation Lookaside Buffer can be use to overcome the problem.

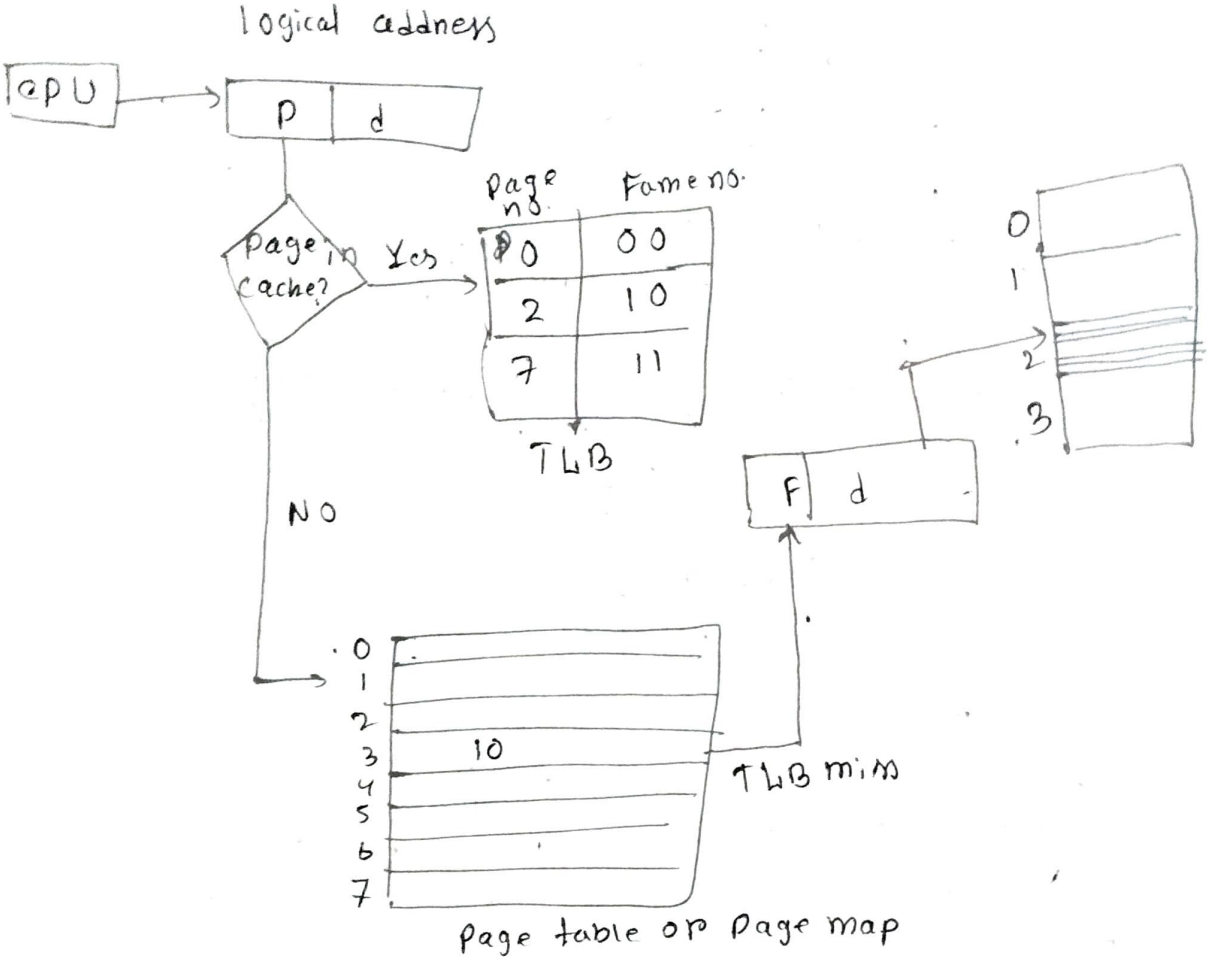


Figure: TLB operation

TLB can be defined as a memory cache which can be ~~defined~~ used to reduce the time taken to access the page table again and again. It's a memory cache which is closer to CPU and the the time taken by CPU to access TLB is lesser then ~~to~~ taken to main memory.

3(b) question Answer

(i)

Need Matrix =

A	B	C	D
1	2	0	2
2	1	0	0
0	3	0	0
0	1	1	1

(∵ Need = Max − Allocation)

From table-2 we Found:

P_2 (1100), P_1 (1,2,00)

For P_2

1) request ≤ need

1100 ≤ 2100

2) request ≤ Available

1100 ≤ 2311

3) Available = Available − request

= 2311 − 1100

= 1211

Allocation = Allocation + request

= 0110 + 1100

= 1210

need = need − request

= 2100 − 1100

= 1000

4) Safety algorithm:

Work = Available

Work = 1 2 1 1

F[1] = F
F[2] = F
F[3] = F
F[4] = F

P_1 = need ≤ work

= 1 2 0 2 ≤ 1 2 1 1

= False

P_2 = 2 1 0 0 ≤ 1 2 1 1

= ~~True~~ = ~~False~~ = ~~True~~ = True

- Work = work + Allocation

= 1 2 1 1 + 0 1 1 0

= 1 3 2 1

P_3 = 0 3 0 0 ≤ 1 3 2 1

= False

Work = 1 0 1 1 + 1 3 2 1

= ~~2 3 3 2~~ 2 5 3 2

P_4 = 0 1 1 1 ≤ 2 5 3 2

Work = ~~3 6 4 2~~ 1 1 1 0 + 2 5 3 2

= 3 6 4 2

Here, 'P_1' process is false and other process are true. So, the request are not granted.

In case OS finds any of the dead locks then it will recover the system using some recovery techniques.