ASSIGNMENT

Course Code : CSE 323

Course Title : Operating System

Topic : SPRING- 2024 Question solve

Submitted to:

Faiza Feroz

Lecturer

Department of CSE.

Daffodil International University

Submitted by:

Name: **Tanvir Ahammed Bipul**

ID: 221-15-4925

Section: 61_J

Department: CSE

Daffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY

Ans: to the Ques no: 1 (a)

To solve the question based on the Bankers Algorithm:
Given,

Total instances: A=14, B=10, C=13, D=12

(I)

Process	Allocation				Maximum				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	2	3	0	1	4	5	0	1	2	2	0	7	5	8	10
P_1	3	1	0	1	5	2	0	2	2	1	0	1	7	7	11	10
P_2	1	0	1	0	1	2	2	2	0	2	0	2	10	8	11	11
P_3	0	0	0	1	0	1	0	1	0	1	0	0	11	8	12	11
P_4	2	1	1	0	3	2	3	1	1	1	2	1	11	8	12	12
P_5	1	1	0	0	2	2	1	1	1	1	1	1	13	9	13	12
	7	5	5	2									14	10	13	12

Here,
Need = Maximum − Allocation

∴ safe sequence: $P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5$ Ans:

(II)

Given Request are:

$P_0 (13, 5, 3, 1)$

$P_1 (0, 1, 0, 0)$

Now we have to calculate is those request are immediate granted or not;

conditions for granting a Request:-

one request from process P_0 for (13, 5, 3, 1):-

1. Request $\leq$ need!

$$Request[P_0] \nleq Need[P_0]$$

- $Need[P_0] = (1, 2, 2, 0)$
- $Request[P_0] = (13, 5, 3, 1)$

For this condition request not satisfied.

2. Request $\leq$ Available!

$$Request[P_0] \nleq Available[P_0]$$

- $Available[P_0] = (7, 5, 8, 10)$
- $Request[P_0] = (13, 5, 3, 1)$

Also for this condition request not satisfied.

Now another request from P_1 (0, 1, 0, 0):-

where both condition are satisfied.

1. Request $\leq$ need!

$$Request[P_1] \leq Need[P_1]$$

- $Need[P_1] = (2, 1, 0, 1)$
- $Request[P_1] = (0, 1, 0, 0)$

2. Request $\leq$ Available!

$$Request[P_1] \leq Available[P_1]$$

- $Available[P_1] = (7, 5, 8, 10)$
- $Request[P_1] = (0, 1, 0, 0)$

Now we have to calculate safe sequence for process P_1 using Bankers Algorithm:-

Process	Allocation				Maximum				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	2	3	0	1	4	5	0	1	2	2	0	7	4	8	10
P_1	3	2	0	1	5	2	0	2	2	0	0	1	7	6	11	10
P_2	1	0	1	0	1	2	1	2	0	2	0	2	10	8	11	11
P_3	0	0	0	1	0	1	0	1	0	1	0	0	11	8	12	11
P_4	2	1	1	0	3	2	3	1	1	1	2	1	11	8	12	12
P_5	1	1	0	0	2	2	1	1	1	1	1	1	13	9	13	12
	7	6	5	2									14	10	13	12

safe sequence: $P_0 \to P_1 \to P_2 \to P_3 \to P_4 \to P_5$ Ans!

So, here for process P_1. It will be granted immediately.

Ans: to the ques no: 1(b)

Given that,

Resource nodes:

- R_1(Laptops) = 3 instances
- R_2(Routers) = 2 Instances
- R_3(Hard Drives) = 2 instances

Process nodes; A, B, C, D

Now we have to create a Resource Allocation graph from given instances & process:-

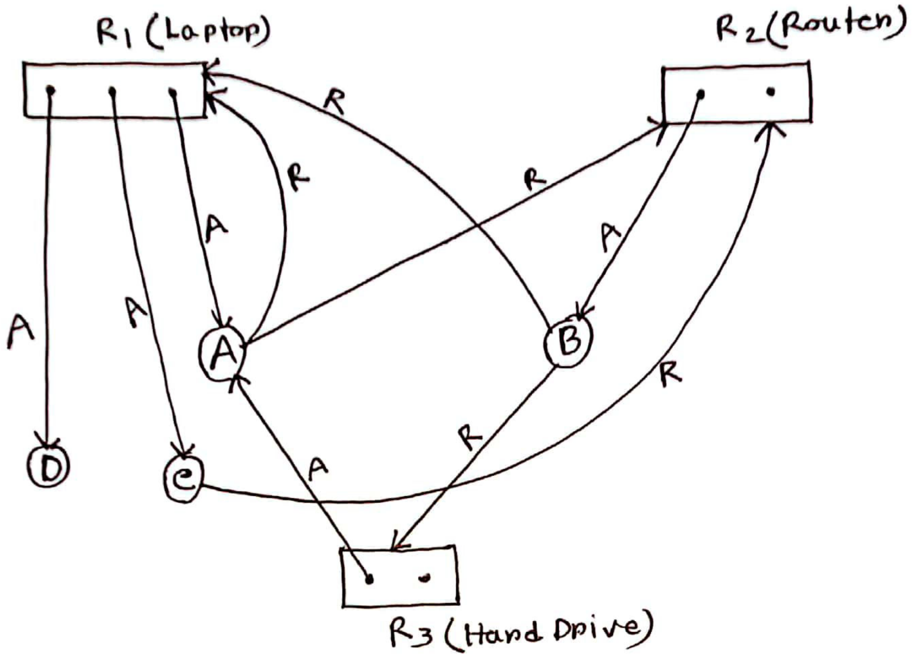

Figure: Resource Allocation graph

Ans: to the Ques no: 1 (c)

Given Resources are $R_0, R_1, R_2, R_3, R_4, R_5$. Which have multiple instances. And each of the process was on assigned by those resources or requested to resources.

Now, If all processes can complete, the system is safe, and we provide the safe sequence.

If any process cannot complete due to insufficient resources, the system is unsafe.

so, At first we can start from any of the process where he can completed.

<u>steps</u>|

1. <u>P_0 Process</u>: Here P_0 is assigned one instance from R_1 and requested one more instance from R_2. so both condition are satisfied, for thats why the P_0 process was complete and freed up the resources.

2. <u>P_2 Process</u>| Here P_2 is assigned one instance from R_0 and request another instanee to R_0. And also request another from R_3. where the condition are fulfilled so the process are complete and freed up resources.

3. This way the processes will complete. How we see the completion in table.

Process	Allocate R_0	R_1	R_2	R_3	R_4	Request R_0	R_1	R_2	R_3	R_4
P_0	0	1	0	0	0	0	0	1	0	0
P_1	0	0	0	0	1	1	0	1	0	0
P_2	1	0	0	0	0	1	0	0	1	0
P_3	1	0	1	0	1	0	0	0	0	1
P_4	0	0	0	0	0	0	0	1	1	0
P_5	0	0	0	1	0	0	0	0	0	0

current Available

R_0	R_1	R_2	R_3	R_4
2	0	1	1	0
0	1	0	0	0
2	1	1	1	0
0	0	0	0	1
2	1	1	1	1
1	0	0	0	0
3	1	1	1	1
1	0	1	0	1
4	1	2	1	2
0	0	0	0	0
4	1	2	1	2
0	0	0	1	0
4	1	2	2	2

safe sequence: $P_0 \to P_1 \to P_2 \to P_3 \to P_4 \to P_5$ <u>Ans</u>!

∴ so, the state is safe.

Ans: to the Ques no: 2 (a)

Dynamic partitioning or variable sized partitioning is a memory management scheme where the memory is divided into partitions based on the sizes of incoming processes.

Given,

Total physical memory: 8 GB

Partition 1: 2 GB

Partition 2: 3 GB

Partition 3: 1 GB

unallocatted: 8GB-(2+3+1)GB = 2 GB

In dynamic partitioning it works with 4 types of partitioning:

1. First fit:

Suppose some process will arrive in first fit partitioning with some sizes then it partitions like;

process

$P_1 \to 1$

$P_2 \to 1.5$

$P_3 \to 2$

$P_4 \to 2.5$

Partition	Memory size (GB)	Process (GB)
1	2→1	P_1
2	3→1.5	P_2
3	1	External fragmentation
Un-Allocated	2 GB	////

2. Next fit: In next fit it will take process after next allocation memory.

Partition	Memory	Process
1	2 → 1	P_1
2	3 → 1.5	P_2
3	1	External f.
Un-Allocated	2	////

Process

$P_1 \rightarrow 1$

$P_2 \rightarrow 1.5$

$P_3 \rightarrow 2$

$P_4 \rightarrow 2.5$

3. Best fit:

where the memory sized is available it will put the process. where minimum waste

Partition	Memory	Process
1	2 → 1	P_2
2	3 → 1.5	P_3
3	1 → 0.5	P_1
Un-Allocated	2	////

Process

$P_1 \rightarrow 0.5$

$P_2 \rightarrow 1$

$P_3 \rightarrow 1.5$

$P_4 \rightarrow 2$

4. Worst fit!

where the memory is in the most high size, then here the process will completed.

Partition	Memory (GB)	Process
1	2 → 0.5	P_3
2	3 → 2.5 → 1.5	P_1, P_2
3	1	P_4
Un-Allocated	2	/////

Process

$P_1 \rightarrow 0.5$

$P_2 \rightarrow 1$

$P_3 \rightarrow 1.5$

$P_4 \rightarrow 1$

so, As this way the Dynamic partitioning are works.

Ans: to the ques no: 2 (b)

In real-life scenarios, the performance of computing devices can vary due to how efficiently memory management techniques are implemented.

1. Insufficient Physical Memory (RAM):

When the devices RAM is insufficient to handle all active processes, it leads to thrashing, where the system spends more time swapping data between RAM and the disk rather than executing tasks.

Paging: Divides memory into fixed-size pages and maps them to frames in physical memory.

Page replacement Algorithms:

- LRU: Replace the page that has not been used for the longest time.
- FIFO: Replaces the oldest page. Simple but may lead to suboptimal performance.

2. Fragmentation

Fragmentation occurs when memory is poorly managed, leading to either:

- Internal Fragmentation
- External Fragmentation

Dynamic partitioning: Allocates memory based on process size, but it can lead to external fragmentation.

Algorithm:

Best Fit: Allocates the smallest suitable block. Reduces internal fragmentation but increases external fragmentation.

3. Poorly optimized virtual memory usage:

virtual memory allows a system to use disk space as an extention of RAM. Excessive reliance on virtual memory slows down the system.

Demand paging: Loads pages into memory only when they are required. overuse of virtual memory can lead to page faults.

optimal page replacement Algorithm: Replaces the page that will not be used for the longest time in the future

4. Deadlock or Resource starvation:-

process may enter a state of Deadlock if they are waiting for memory or other resources Indefinitely.

Deadlock Avoidance: Allocates resources such that circullar wait is avoided.

- Bankers Algorithm: Predicts whether a resource allocation will leave the system In a safe state. prevents deadlock but may delay processes.

Ans: to the ques no/ 3(a)

To analyze the page replacement sequence for the given page reference string using LRU and optimal algorithms, we need to simulate the behaviour of each algorithm step by step.

Simulation for LRU Algorithm :-

page	1	2	3	4	1	2	5	1	2	3	4	5	6	2	1	3	7	7	0	9
F_1	1	1	1	4	4	4	5	5	5	3	3	3	6	6	6	3	3	3	~~0~~3	9
F_2		2	2	2	1	1	1	1	1	1	4	4	4	2	2	2	7	7	7	7
F_3			3	3	3	2	2	2	2	2	2	5	5	5	1	1	1	1	0	0
	F	F	F	F	F	F	F	H	H	F	F	F	F	F	F	F	F	H	F	F

∴ Total page Faults (LRU): 17

simulation for optimal Algorithm :

page	1	2	3	4	1	2	5	1	2	3	4	5	6	2	1	3	7	7	0	9
F_1	1	1	1	1	1	1	1	1	1	3	4	4	6	6	1	3	7	7	0	9
F_2		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
F_3			3	4	4	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	F	F	F	F	H	H	F	H	H	F	F	H	F	H	F	F	F	H	F	F

∴ Total page Faults (optimal): 13

The optimal algorithm performs better than LRU because it has prior knowledge of future references, allowing it to minimize page faults effectively.

Ans.

Ans: to the ques no| 3(b)

Here are the general strategies to enhance both LRU and optimal Algorithm:-

1. Increase physical memory: Adding more physical memory reduces the need for frequent page replacements, benefiting both algorithms.

2. Reduce working set size: optimize applications to minimize their working set, reducing page replacement frequency.

3. Use Multi-Level caches: Implement multi-level caching to handle most memory requests without frequent page faults.

4. Optimize virtual memory management:
   - Adjust the page size to match typical workload patterns.
   - use prefetching to load pages into memory in anticipation of future requests.

5. Monitor and Adapt workload Behaviour:
use system monitoring tools to detect and adapt to changes In workload patterns dynamically.

Trade-offs and practical constraints:-

1. Complexity vs. performance:

- optimal enhancements, such as predictive algorithms, increase computational complexity but can significantly improve performance
- LRU approximations like clock reduce complexity but may occasionally make suboptimal replacements.

2. Hardware vs. software cost:

- Hardware-assisted LRU implementations require specialized hardware, increasing costs.
- Software-only solutions, while cheaper, are computationally intensive.

By implementing these strategies, the efficiency of both LRU and optimal algorithms can be improved, addressing their limitations while balancing performance and resource usage.