*315 exam 2 Review: This is just answers to your questions and is not NOTE ➔

1) Deadlock avoidance / Banker's alg / Safety alg & Detection alg

replacement for your Readings!

2) Difference bw scheduling techniques & how they work

3) strategy for avoiding circular wait

4) Deadlock Outline map. Deadlock → avoidance, prevention, Detection

5) How to implement preemption?

6) what is a trap?

7) Optimal / FIFO / LRU

8) Resource allocation Graphs

9) Segmentation vs Fragmentation

10) example of paging & binding physical memory addr from logical. ← paging activity

11) Difference bw segmentation & paging

12) VM → switching pages around ?!?

Deadlock

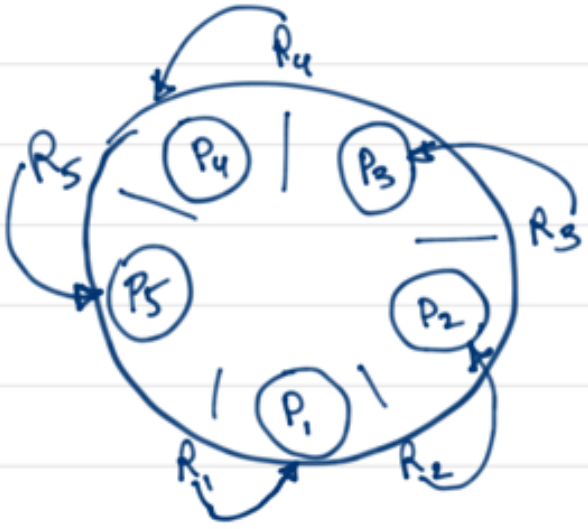
- In a multiprogramming environment, processes compete for resources. if a process i requests a resource that is being held by another process j that is waiting on another resource held by process i

  ⇒ Deadlock

- Conditions for a deadlock? The following 4 should hold simultaneously
  1. Mutual Exclusion
  2. Hold and wait: a process is holding at least one resource and waiting to acquire a resource that is being held by another process
  3. No Preemption: resources can be only released voluntarily
  4. Circular wait: A set of waiting processes $\{P_0, P_1, \ldots, P_n\}$ of waiting processes must exist such that P_0 is waiting on a resource held by P_1 ... P_n is waiting for a resource held by P_0.

Dining Philosophers

1. ME
2. Hold & Wait
3. Circular wait
4. No Preemption

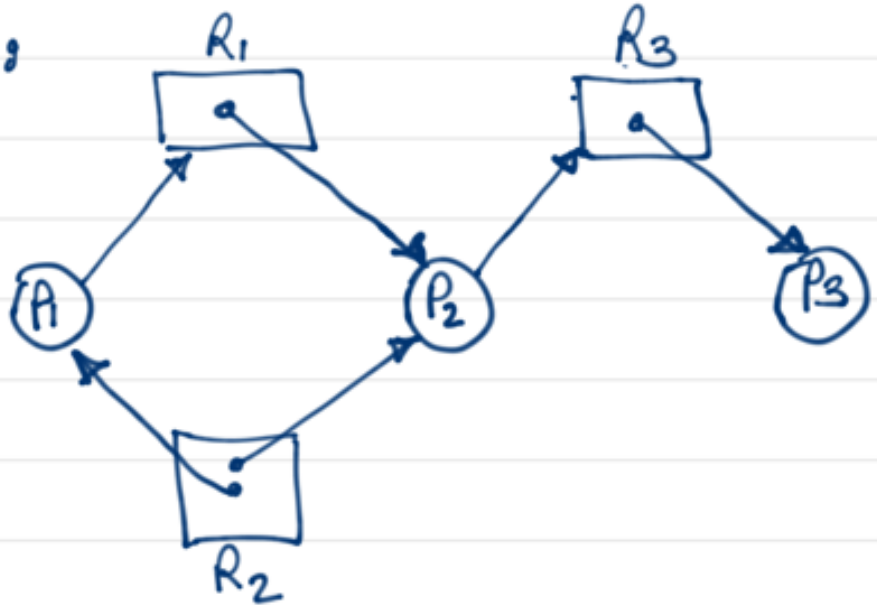


Models of system resource state:

1) Resource – Allocation Graph: nodes: Processes & Resources

edges: Resource → Process: allocation edge
Process → Resource: Request edge

example:



2) State of the resource-allocation system (Matrix Notation) vector +
for m resource types + n processes

Available, number of available resource (vector size m)

Max: n x m matrix, maximum demand of each process.

Allocation: n x m matrix, number of resources of each type allocated to each process.

Need = Max - Allocation

*How to handle deadlock?

1) avoid or prevent (so we never enter a deadlock situation)

2) allow deadlock & then detect & recover

3) Ignore it!

1) avoid or prevent:

1-A) Deadlock Prevention: Ensure at least one of the 4 Condition does not hald by some protocols

1-A-i) Mutual Exclusion: not always possible

1-A-ii) Hold & wait: not allow a process which is holding a resource to request new resource (a) request all resources at begining (b) release all resources before new request

1-A-iii) No Preemption: (a) if a process request some resource

that is not available ⇒ all resources the process is holding currently is preempted. (b) when a process request a resource that is held by some other process, we check if other process is waiting on another resource → preempt the resource.

1-A-iv) Circular Wait: apply some ordering on how resources can be requested. Dining Philosophers: increasing order.

1-B) Deadlock Avoidance: Based on additional info about how resources are to be requested. (Assumes that OS knows what resources will be requested by each process).

+ Safe State: Define a safe state where system can allocate resources to each process (up to its maximum) in "some order" and avoid a deadlock.

Avoidance algorithms:

Request → Safe(?) Yes → Grant

No → Don't grant

1-B-i) Resource allocation: Possible Future requests: claim edge
→ after each request → Safe? (using cycle detection)
(yes → allocate
No → don't allocate

1-B-ii) Banker's Algorithm: Before granting each request, evaluate and see if the system stays in a safe state after this allocation or not → Yes → Allocate; No → don't allocate.

Safe state: Considers the worst case scenario, where all processes request all their needed resources & see if there is <u>one</u> sequence of execution or not.

Yes → safe
No → unsafe

example:

	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	7	4	3 ✓	3	3	2
P_1	2	0	0	3	2	2	1	2	2 ✓	5	3	2
P_2	3	0	2	9	0	2	6	0	0 ✓	7	4	3
P_3	2	1	1	2	2	2	0	1	1 ✓	7	4	5
P_4	0	0	2	4	3	3	4	3	1 ✓	10	4	7

System currently in a

$(P_1, P_3, P_4, P_2, P_0) \Rightarrow$ Safe State

1) Can a request (1,0,2) be granted? (Available: 3,3,2)
Yes safe

2) Can a request (0,2,0) by P_0 be granted? after previous request?

P_1: (1,0,2)

	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	7	4	3	2	3	0
P_1	3	0	2	3	2	2	0	2	0			
P_2	3	0	2	9	0	2	6	0	C			
P_3	2	1	1	2	2	2	0	1	1			
P_4	0	0	2	4	3	3	4	3	1			

Safe (P_1, P_3, P_4, P_0, P_2)

P_0: (0,2,0)

	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	0	3	0	7	5	3	7	2	3	2	1	0
P_1	3	0	2	3	2	2	0	2	0			
P_2	3	0	2	9	0	2	6	0	C			
P_3	2	1	1	2	2	2	0	1	1			
P_4	0	0	2	4	3	3	4	3	1			

Safe? No ⇒ Revert back

2) Deadlock Detection & Recovery:

Detection (A)

2-A-i) Cycles in a Resource allocation graph.

Wait-for-graph : all for single resource

2-A-ii) Deadlock detection algorithm (Similar to Banker's)

[Available, Allocation, Request] } if there is a sequence of execution for the current set of requests → Yes → No deadlock

→ No → Deadlock

Key Question: what is the difference bw this alg & Bankers?

2-B.: Deadlock Recovery: How to recover from it?

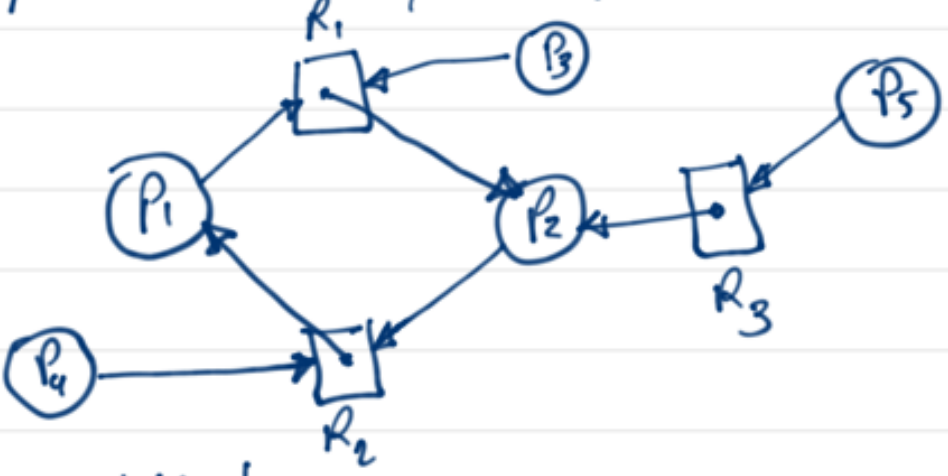
2-B-i) Process termination : (a) all processes : Cost?

(b) one by one : Cost?

2-B-ii) Resource Preemption : First choose a victim process based on some cost, second : roll-back the process to a safe state (or abort & restart), third watch out for starvation how?

Key Questions:

1) How can we detect a deadlock? (single vs multi resource)
2) What happens to the system if a deadlock happens?



Can grow very quickly!

3) How can we prevent a deadlock?

4) if we have apriori info about all processes & their future requests, can we avoid deadlock? How? (single resource vs. multiple)

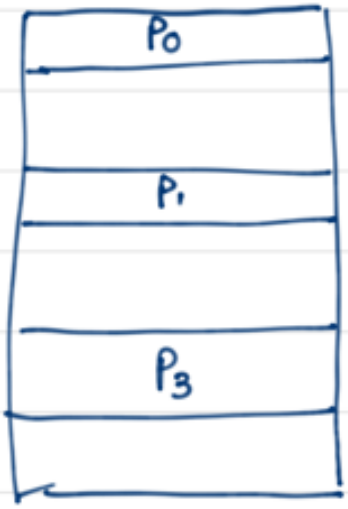
CPU Scheduling

1) FCFS
2) SJF (Preemptive & non-preemptive)
3) RR (with diff quantum size)
4) Priority (Preemptive & non-preemptive)

- Multilevel feedback queue : diff schedulers for diff queues.

Memory Management:

Each Computer System has a limited amount of memory which should be shared among different processes. Processes enter/exit the system & hence the need for a manager to keep track of & manage the memory space is crucial.



1) Contiguous Memory Allocation:
   - 1-A: Best fit
   - 1-B: First fit
   - 1-C: Worst fit

2) Segmentation:
   decrease external fragmentation by breaking down each process address space into smaller pieces (segments).

3) Paging: Divide process's addr space into equally sized pieces called pages and fit those into free frames available throughout the memory.
   Very powerful technique that can helps us to implement virtual memory.

Virtual Memory:

- There is no need to load the whole addr space into memory ⇒ Why not just load the pages that are being accessed by the process (Demand Paging)?
- Also, what if a process is larger than the physical memory?
- Higher degree of multiprogramming ⇒ Higher CPU utilization, Higher throughput

⇒ OS always want to achieve the highest throughput ⇒ if the utilization is low ⇒ increase the degree of multiprogramming

⇒ What can go wrong? ⇒ high page fault rate ⇒ Long queue for Backing store ⇒ Low CPU utilization

⇒ Thrashing: process is busy swapping pages in & out!

Why VM worked? ↑ if swapping too much ⇒

⇒ How much frames shall we allocate each process?

Global vs. Local allocation

Priority ↵

↳ result in Thrashing where more processes in memory than optimal.

↳ Fixed allocation

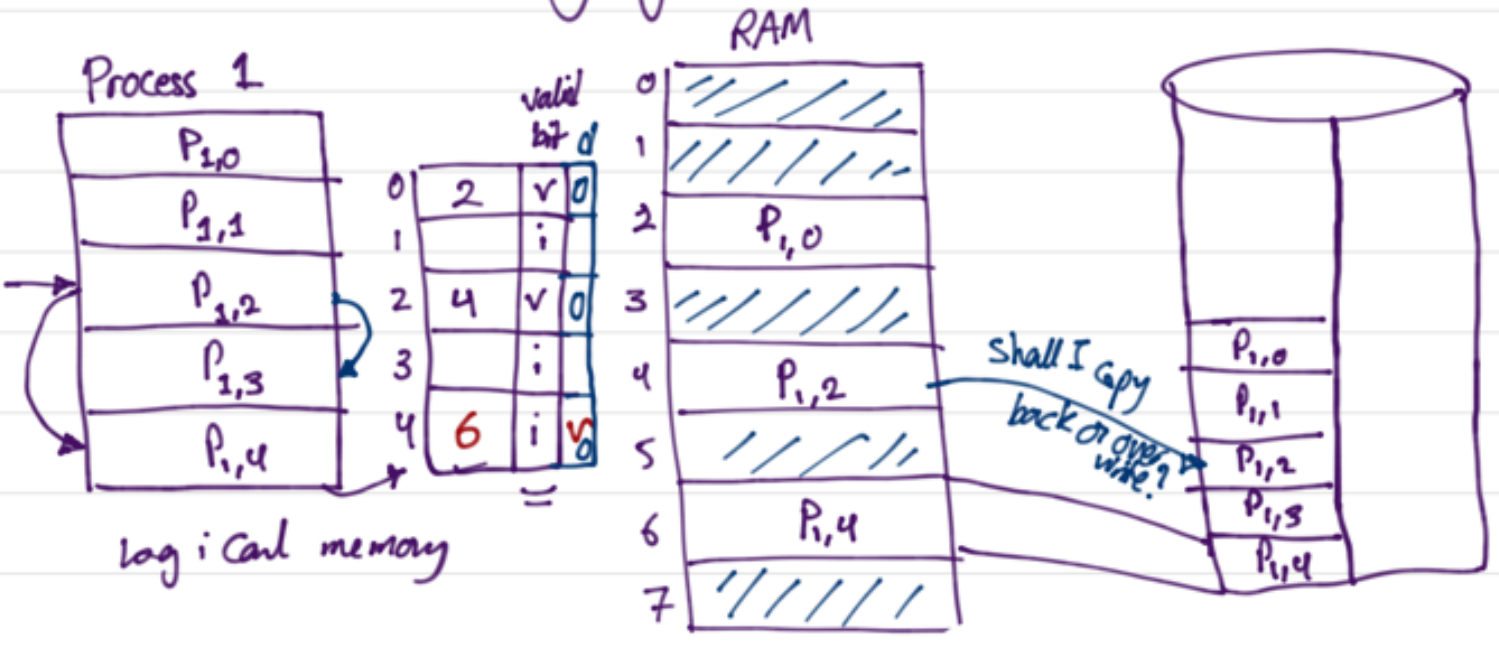
⇓

One process paging in & out ⇒ swap device busy ⇐ Slow the system

* What was a page fault?

No Free Frame!

Process is running & it tries to access a page that is not loaded into memory yet.



Probability of page fault: $0 \leq p \leq 1$ ⟶ hit $(1-p)$ 8,000,000 nsec

Effective access time? Page fault overhead = $\Delta = 8$ msec

memory access = $\delta = 200$ nsec

Huge gap

$EAT = (1-p)\,\delta + p\,\Delta$ if $p = 0.001$ 0.1%

EAT 8200ns → 41 times slower than memory access

acceptable EAT ⟹ way lower p

Why does this work? Locality ← 1 every 400,000 time!

Virtual Memory:

Now that we have the underlying paging mechanism, we can take advantage of that and separate the process view from the actual limitations of Physical Memory.

how many 512 MB processes? 16 → if just use 128MB for each process? 80 processes!

⇓

higher throughput, higher CPU utilization

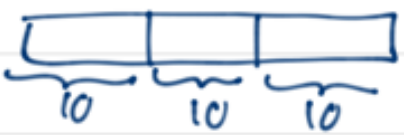
As long as the process has access to what it needs for execution ⇒ it does not have to know what is going on at low level.

1) We don't need to load the whole process

2) Using page fault mechanism we can give the process illusion of having all its logical addr space.

System with 44 bits logical addr has 16GB RAM. How many bits necessary to address all frames in RAM? Show the Mapping from logical to physical by MMU if page size is 16KB. (2^{14}) ← 2^{34}

- How big can a page table be? → Make sense? How can we fit it into RAM?

2^{30} page table : 1GB : Hierarchical 

Or 10 and 20.

- How many frames should we allocate to each process?
  1) proportional to its size
  2) . . . priority