Final Exam

Q1 - Deadlock

Deadlock Conditions - 4

DEADLOCK □ ALL 4 TRUE

- Mutual exclusion - resourced used by 1 process @ time
- Hold + wait - proc w resources waiting for another proc's resources
- No preemption - resources ONLY voluntarily released
- Circular wait - each process waiting for next + last wait for first
  - *Circular wait implies hold + wait*

Handling Strategies - 3

PREVENT - ALLOW - IGNORE

- Ensure deadlock never happens via avoidance + prevention techniques
- Allow deadlock then recover via detection + recovery techniques
- Ignore the problem

Prevention Methods - 4

- Deny mutual exclusion - allow resource sharing
  - Con - some resources not sharable e.g. printer
- Deny hold and wait - only request when **NOT** holding resources
  - Con - poor resource utilisation + starvation
- Allow preemption - all resources can be taken away from proc.
  - Only works for state saveable resources
- Deny circular wait - resources ordered + requested in increasing order
  - Potentially impossible to find order that satisfies every process

Deadlock Avoidance

- System needs future info about resources needed by processes
- Needs to know:
  - Max - possible resources
  - Available - currently

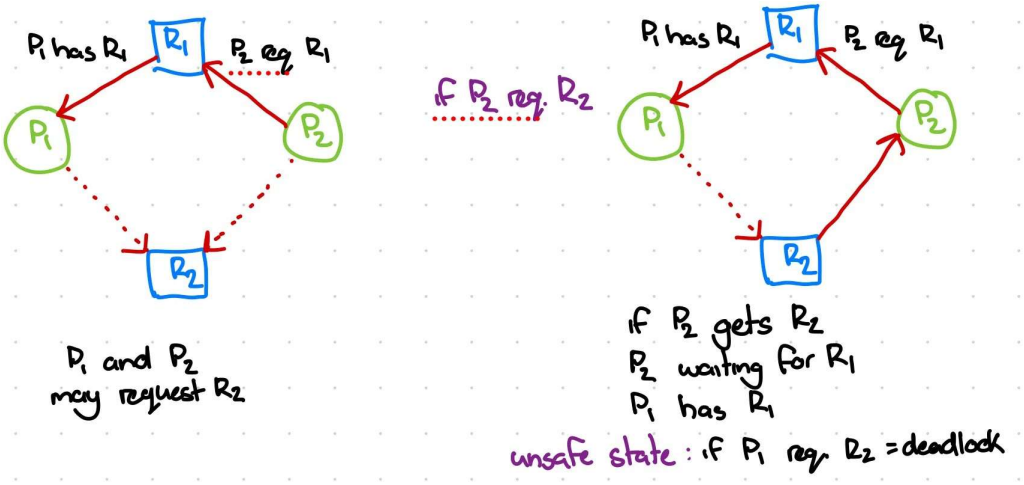
- Allocated - to process currently

Recovery Methods - 2

- Kill processes
  - Problem - process in middle of important operation
    - Solution - save resource state before killing
  - Based kill off priority || resources used || resources needed
- Preempt resources
  - (Process) rollback → safe state + restart
  - Problem - low priority always killed - uses resources + makes no progress
    - Solution - include # rollbacks in victim choice

Resource Allocation Graph

- Cycle in graph:
  - Only 1 of each resource = deadlock
  - Multiple of each resource = potential deadlock
- Wait-For graph = resource-allocation graph but remove resources - *ONLY PROCESSES*
  - ***ONLY USED WHEN SINGLE INSTANCE RESOURCES***



Bankers Algorithm

- If request > need - DENY
- If request > available - DENY
- If request < need && request < available → check safe state

Matrixes

- need = max - allocated
- Process (P1) finish → available = previousAvailable + P1allocated
    - Repeat for all process

Q2 - Memory Management

Fragmentation

- External - memory available BUT in holes too small to satisfy request
- Internal - allocated > requested = wasted
    - *Internal to a partition*

OS Responsibilities

- Track used memory
- Choose next loaded process
- Allocate + deallocate mem

Paging System

- Memory divided into fixed-size pages
- Fixed size = internal fragmentation IF allocated > requested size
- Page - virtual memory block
- Frame - physical memory block
- Frame size == page size
- Process access page - MMU translate → frame
- Formulas:
    - # of frames = phys @ size / frame size
    - # of pages = virtual @ size / page size

Page vs Segment

- Pages - fixed length
- Segment - varying length
    - Logical @ split into groups
    - Has name + length
    - Access by - name + offset (within segment)
    - STBR (seg table base reg) - points to seg table location
    - STLR (seg table length reg) - # of segments used by prog