OS SHORT NOTES FOR UNIT-3
Ishan Dubey

- Deadlocks
    - A set of processes are in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set.
    - eg. Dining Philosopher's Problem
        - Possible solution: initialize n semaphores to 1
        - Possible solution: do{ wait(chopstick[i+x%n]); ...eat...; signal(chopstick[i+x%n]);...think...} while(1)
        - Here x ranges from 0 to n
    - Necessary conditions for deadlock
        - Mutual exclusion of resources: At least one non sharable resource
        - Hold and wait: A process must be holding one resource and waiting to acquire other resources held by other processes
        - No preemption: RESOURCES cannot be preempted.
        - Circular wait: Pi waiting for P(i+1)%n
    - Resource allocation graph
        - No cycle -> No deadlock
        - Deadlock ->Cycle
        - Cycle->Deadlock case when each resource has exactly one instance
    - Handling Deadlocks
        - Prevention: Take away any of the necessary conditions
            - Mutual exclusion: Can't take this away cuz some resources are intrinsically non-sharable. eg. printer
            - Hold and wait: Ask processes to request resources ar startup or to request when it's holding nothing. Inefficient. Starvation
            - No preemption: Either preempt resources held by waiting process, or preempt the needed ones from others. Difficult for I/O devices
            - Circular wait: Enumerate the resources and enforce an order. Inflexibility for programmers
        - Avoidance: Concept of safe and unsafe states
            - If resource granting is safe, grant it. Else, process waits
            - Single instance of resources: Resource allocation graph algo
                - See if granting the request may cause a deadlock (cycle)
                - If so, don't grant the request
            - Multiple instances: Banker's algo
                - Allocation: Current allocation of resources. n*m matrix
                - Max: TOTAL MAX no of resources each process needs to finish. n*m matrix
                - Need: Max - Allocation. n*m matrix
                - Available: How many resources are available. m vector
        - Detect and Recover: Provide and algorithm
            - Process termination: Terminate ALL processes in the deadlock. Bad. Abort one at a time. Requires policy to select
            - Resource preemption one at a time
        - Ignore: Ostrich approach. Best
- Memory Management

- Logical address space: Base and Limit registers. Last register = Base + Limit
- Hardware access protection from CPU----->Memory. Check above condition.
- Address binding: Take a process from input queue (on disk) and put it anywhere in the memory
    - Compile time: If where process will reside in memory is known, generate absolute code.
    - Load time: If it's not known, generate relocatable code.
    - Execution time: Delay binding till runtime. Special hardware needed.
- See multistep processing of a user program from PPT
- Address space
    - Logical: Generated by CPU. Virtual address
    - Physical: As seen in memory unit
    - The spaces differ in execution time address binding.
    - Mapping is done by Memory Management Unit (MMU) and base register is now called relocation register
- Dynamic loading: Don't load a routine if it's not called. If it's called, check if it's already been loaded first. If not, call relocatable linking loader to do it. No special support from OS
- Dynamic linking: Include stub in program for each library routine reference telling how to locate or load the routine. Needs help from OS.
- Swapping
    - Swap the process to a backing store and bring it back later
    - If used for priority-based scheduling, it's called Roll Out, Roll In
    - Do reasonable computation between swaps. (Large enough time quantum)
    - Where will the process be brought back?
        - If load/assembly time binding, then at the same place
        - Execution time binding: Anywhere in the memory as physical addresses are calculated at execution time.
    - See diagram from PPT
    - Constraints: High context switch time incl. swapping. Can't do if pending I/O. Double buffering adds overhead. Swapping disabled by default
- Allocation
    - Contiguous: Two partitions of memory. OS (low), User processes(high). Use relocation registers to separate, and MMU maps logical address dynamically
    - Multiple-partition
        - First fit, best fit, worst fit. Worst fit worst
- Fragmentation
    - External: Enough space but not contiguous. Solve by compaction (reshuffling) or paging
    - Internal: Lots of small holes
- Segmentation
    - User view of memory. Functions, objects, stacks, arrays, etc. Logical units
    - Segment table containing base and limit registers for segments
    - Segment table base register (STBR): Base register for segment table
    - Segment table length register (STLR): Number of segments
    - See segmentation hardware diagram from ppt
- Paging (Solves external fragmentation problem)
    - Physical memory divided into frames

- Process divided into pages. Pages sit in frames
- IMP: Logical address: m-p bit page number, p bit page offset. Page size is 2^p and Logical address space is 2^m
- Page table in main memory. PTBR and PTLR just like segmentation
- Use Translation Lookaside buffer for faster access
- EAT = hit_ratio(time to access memory once to get there) + miss_ratio(time to access memory to get page table + time to access memory again to get there) = a*(1+e) + (1-a)*(2+e)
- Valid-Invalid bit is used

- Paging Methods
  - Hierarchical
    - Two-Level scheme: 12+10+10 bits
    - Three-level scheme: 42+10+12 bits
    - Four-level scheme: 32+10+10+12 bits
  - Hashed
    - Hash the page table. Useful for sparse address spaces
  - Inverted
    - Map frame to pages instead of pages to frame
    - Only one page table needed in the system
    - Difficult to implement shared-memory and lookup is slower unless hashed

- Virtual Memory
  - Definition: Separation of User logical memory from physical memory
  - Can load part of process in the memory. Address space can be shared by several processes. Allows wayyy larger logical address space
  - Virtual address space: Stack grows down from max, heap grows up from min. Share pages also.
  - Demand paging (Paging + lazy swapper)
    - Bring a page into memory only when it's needed
    - Less I/O, less memory, faster response, more user
    - Handling page faults
      - 1. Page not in memory is referenced
      - 2. Leads to a trap in OS
      - 3. Page is in the backing store
      - 4. Bring the page to physical memory in a free frame
      - 5. Reset page table
      - 6. Restart the instruction that caused the page fault
    - EAT = (page_fault_rate)*(page_fault_time) + (1 - page_fault_rate)*(memory_access_time)
  - Page Replacement (to prevent over-allocation of memory)
    - Basic algo
      - 1. Find desired page on disk
      - 2. Swap out victim and update page table
      - 3. Swap in desired page and update page table
      - 4. Restart user process
    - FIFO: Use a queue
    - Optimal: Replace page that will not be used for longest time. Best but also hardest to implement as you need to know reference string

- LRU: Replace least recently used page. Counter and stack implementations
- LRU and Optimal are stack algos as they don't show Belady's anomaly
- Belady's anomaly: Page faults increasing as no. of frames are increased
- LRU Approximations: Reference bit (replace page with referenced = 0 if exists), Second-chance (FIFO + Referenced bit)
- Least Frequently Used (LFU)
- Most Frequently Used (MFU)

- Thrashing
  - Page fault at almost every reference if process doesn't have enough pages. So much time spent doing only swapping
  - Cause
    - 1. CPU utilization low, so we increase degree of multiprogramming by adding a process
    - 2. Global page replacement algo replaces pages INDISCRIMINATELY
    - 3. Upon faulting process steals frames from other processes, causing low CPU utilization. Loop back to step 1 and keep worsening the effects.
  - Can be prevented by using local page replacement algo or priority page replacement algo, but it's not a complete solution
  - To prevent, provide a process with as many frames it needs