☺***OS Final Exam Revision Short Notes***☺

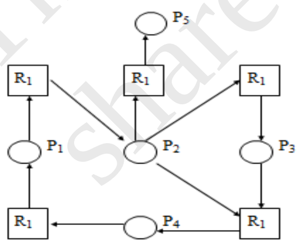
Lecture 4 – Concurrent Processes

- Shared logical address space. Each operation must be executed *atomically*.
- **Race condition**–
    - Outcome depends on **order of execution**. (Several processes access and manipulate same data concurrently)
    - To prevent race condition we need **process synchronization**.
- **Critical section problem** – Code segment with shared data. Only one process should be allowed to change variables.
    - Make sure when one process executes the critical section**, others are not allowed**.
    - **Solution** –
        - Mutual Exclusion (Only one process can execute in the critical section at a time)
        - Progress (Process outside the critical section cannot block another process)
        - Bounded Waiting (No process must wait forever to enter the critical section)
    - Simplest solution – Process disable interrupts after entering its critical section.
        - Not good because it's a privileged instruction.
        - Not feasible in a multiprocessor system.
    - Solution for two processes
        - Algorithm 1 – Satisfy mutual exclusion but no progress. Busy waiting.
        - Algorithm 2 - Satisfy mutual exclusion but violates progress requirement.
        - Algorithm 3 – Combine shared variables of Alg1 & Alg2. Solves CS problem for 2 processes.
    - Solutions for n processes
        - **Bakery Algorithm** – Receives a number. Smallest no. enters the CS. If 2 processes get same no., smaller PID goes first.
        - **Synchronization Hardware –**
            - Test-and-Set, Swap. (Both don't satisfy the bounded waiting requirement)
        - **Semaphores** – Integer variable that can be accessed only via 2 atomic operations.
            - Two types – Counting semaphore, Binary semaphore.
            - Require busy waiting. **(Spinlock)**
                - <u>Advantage</u> – No context switch needed.
                - <u>Disadvantage</u> – Wastes CPU time.
            - Problems –
                - **Deadlock** (2 or more processes waiting indefinitely for one another)
                - **Starvation** (Indefinite blocking. Never removed from semaphore queue.)
            - Limitation – Can result in timing errors. Can result in deadlock.
        - **Monitors** – Allows only one process to be active in the monitor at a time.
            - x.wait – This process is suspended until another process invokes it.
            - x.signal – Resumes exactly one suspended process. (Different from semaphore.)
- Classical Problems of Synchronization
    - **Bounded-buffer problem**–<u>Mutex</u> is binary, <u>full & empty</u> are counting.
    - **Readers and Writers problem** – If reader has high priority, writers wait.
    - **Dining-philosophers problem**–No two neighbours can eat simultaneously.
- Atomic Transactions – sequence of *read* and *write* operations.
    - Commit, Abort.
    - Abort must have no effect on state of data. (*rolled back* transaction)
    - Log-base recovery
        - Write-ahead log – Transaction name data item name, old value, and new value.
        - Checkpoint – to reduce recovery overhead.

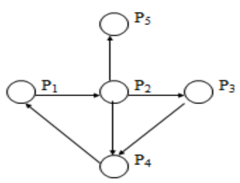
Lecture 5 – Deadlocks

- Several processes compete for finite resources, and some wait forever for the resources held by waiting processes.
- System resources – can be pre-emptable or non-preemptable.
    - Physical resources (printers, tape drives, CPU cycles)
    - Logical resources (files, semaphores, monitors)
- Process requests a resource. (Request, Use, Release)
- Necessary conditions for deadlock
    - Mutual exclusion (Only one process can use the resource at a time)
    - Hold and wait (Process holding a resource is waiting for another resource held by another process)
    - No pre-emption (A resource can only be released voluntarily by the process holding it)
    - Circular wait
- Can be described using *System resource-allocation graph*.
    - If no cycles, no deadlocks.
    - If there is a cycle, a deadlock may exist. (If a resource has many instances, no deadlock)
- Three methods for handling deadlock –
    - Deadlock prevention and/or deadlock avoidance.
    - Deadlock detection and deadlock recovery.
    - Ignore problem. (Assume deadlocks never occur)
- **Deadlock prevention** –
    - Deny mutual exclusion
    - Deny hold and wait
    - Allow pre-emption
    - Deny circular wait
- **Deadlock avoidance** – must have some additional priori information about resource requests.
    - Algorithm dynamically examines the resource-allocation *state*.
    - Safe state – (examined using Maximum-needs, Allocation and Current-Need, and Available)
        - If there is a safe sequence, no deadlock, therefore safe.
    - Resource-Allocation Graph Algorithm (*Claim edge* converts to *request edge* when resource is requested)
    - Banker's Algorithm – for a system with resources with multiple instances.
        - Never allocate a resource if it no longer satisfies requirements of all processes.
        - Time complexity – $O(mn^2)$
        - Considers (Allocation, Max, Available, Need). Check if *request ≤available.*
- **Deadlock detection**
    - Wait-for-graph

Resource Allocation Graph



Wait for Graph



- **Deadlock recovery**
    - Terminate processes. (kill all deadlock processes or kill one at a time until cycle is broken)
    - Pre-empt a resource from a process. (Rollback the process to some safe state and resume from there)
- Combined approach to deadlock handling.
    - Prevention
    - Avoidance
    - Detection

Lecture 6 – Memory Management

- Binding is a mapping from one address space to another.
  - Compile time (Must recompile of starting location is not available)
  - Load time (Must generate relocatable code)
  - Execution time (Need special hardware. **Base and limit registers**)
- **Dynamic loading** – Routine not loaded until called.
- **Dynamic linking** – Used for system libraries. Locates memory-resident using a *Stub*.
- Logical address - generated by the CPU.
- Physical address - address seen by memory unit.
- Run-time mapping from virtual to physical address is done by Memory Management Unit (MMU) – hardware device.
- Swapping – Between main memory and a backing store.
  - Potential problem if swapped out during an I/O operation.
- Contiguous allocation
- Memory Mapping and Protection - done by relocation and limit registers.
  - Relocation register contains smallest physical address.
  - Limit register contains range of logical addresses. Each logical address < limit register.
- Multiple-partition allocation – can be fixed size or variable size partition for each process.
- Dynamic storage-allocation problem – how to satisfy size *n* from available free holes.
  - First-fit (Allocate to 1st hole that is big enough)
  - Best-fit (Allocate to smallest hole available, that is big enough)
  - Worst-fit (Allocate to largest hole available)
- **Fragmentation**
  - External fragmentation – free memory exists but not contiguous.
    - Reduce by *compaction*. But compaction is possible only when relocation is dynamic.
    - Solution- Paging
  - Internal fragmentation – allocated memory (page size) larger than requested memory.
- **Paging** (No external fragmentation but may create internal fragmentation)
  - Logical address space can be noncontiguous.
  - Physical memory is divided into fixed-size blocks, called ***frames***.
  - Logical memory is divided into fixed-size blocks, called ***pages***.
  - Advantage – Shared pages (shared code).
- Logical address – Page # (p) and Page offset (d)
- Physical address – Frame # and offset
- **Page table**
  - Page table base register (PTBR) points to table, Page table limit register (PTLR) shows page table size.
  - This scheme requires 2 memory accesses. (1 for page table access, 1 for data/instruction access)
    - Solution – *associative registers*, aka, *translation look-aside buffers* (TLBs).
- Associative registers
  - If A in associate register, get frame # out. (TLB hit)
  - Else get frame # from page table. (TLB miss)
- **Effective access time** $\underline{\text{EAT} = (t + \varepsilon)\alpha + (2t + \varepsilon)(1-\alpha)}$
- **Memory protection** – valid/invalid bit in page table entry. Valid if page is in the logical address space.
- Page Table Structure
  - Hierarchical Paging (Paging the page table / Two-level paging) More memory access if not in cache.
  - Hashed Page Tables (Commonly used when address space > 32 bits)
  - Inverted Page Tables (Logical address and page table contains pid.)
- **Segmentation** – User-view of memory. Contains segment name and offset (displacement).
  - Segmentation table. (Each table entry has *base* and *limit* registers. STBR & STLR)

- Trap – address error.(if segment no. doesn't exist in segment table)

Lecture 7 – Virtual Memory

- Virtual memory – separation of user logical memory from physical memory. Implemented via,
  - **Demand paging** – Process resides in disk. Bring to memory (not entire process) only when needed.
    - Less I/O, less memory needed, faster response, more users.
    - Hardware requirements – Swap space / backing store, page tables to mark validity.
    - If page not in memory (invalid bit set), generate ***page-fault*** trap.
    - Effective Access Time. **EAT = (1 – *p*) * memory access + *p* * (page fault service time)**.
  - **Demand segmentation** –
    - Used when insufficient hardware to implement demand paging.
    - OS/2 allocates memory in segments, which it keeps track through *segment descriptors* (valid bit).
    - If not in memory, segment fault.
- Other benefits of virtual memory
  - Copy-on-write (Allows both parent and child processes to initially share the same pages)
  - Memory-Mapped Files (Allows file I/O to be from a *mapping* to a disk block to a page in memory)
    - Allows several processes to map to the same file.
- **Page replacement** (Also use a *modify bit*. Only modified pages are written back to disk when replacing.)
  - First-in-first-out (FIFO) algorithm (Suffer from Belady's anomaly, more frames result in more page faults)
  - Optimal algorithm (Replace page that will not be used for the longest time period)
  - Least recently used (LRU) algorithm (Replaces page that has not been used for the longest period)
  - Additional-reference-bits algorithm (8-bit record for each page. Replace page with smallest value)
  - Second chance algorithm (Uses FIFO, but if replaceable page has reference bit 1, set to 0 & try next)
  - Enhanced second-chance algorithm (Uses a modify bit in addition to reference bit)
  - Counting algorithms (Maintains a reference counter. Uses Least Frequently Used (LFU) / MFU algorithms)
- Page Buffering Algorithm (Used along with any replacement alg. OS keeps pool of free frames, uses when page fault)
  - Advantage – No need page replacement.
- Two major allocation schemes –
  - Fixed allocation (*m* frames, *n* processes. $\therefore m/n$ frames per process. Remainder in free frame pool)
    - Proportional allocation (allocate according to the size of process)
  - Priority allocation. (proportional allocation scheme using priorities rather than size)
- Frame replacement – Global replacement (frame from any process), Local replacement (frame from its own process)
- **Thrashing** – Each process is busy swapping, when a process doesn't have enough pages. (Low CPU utilization)
  - When thrashing occurs processes are busy swapping. CPU thinks it's usage is low and introduce new processes.
  - To reduce thrashing, decrease degree of multiprogramming, or use local (or priority) replacement algorithm.
- Prepaging - When reloading a suspended process, load all the working set of pages to prevent many page faults.
- TLB Reach - TLB Reach = (TLB Size) X (Page Size)
  - Increase the size of the TLB
    - **Increase the Page Size -** May increase internal fragmentation.
    - **Provide Multiple Page Sizes.**
- **I/O interlock and addressing** - Problem of a process requesting I/O if I/O is done to/from **user** virtual memory.
  - Solutions
    - Buffer in Operating System space (Time consuming)
    - Lock buffer pages in memory (lock bit is associated with every frame. If lock cannot replace)
      - Problem :Process crashes while page locked - not released., Low priority process waiting.
      - Solution :Prevent replacing a newly brought in page until it can be used at least once.

Lecture 8 – File Concepts

- File – smallest unit of logical secondary storage.
- File system consists of - A collection of files, A directory structure, Partitions.
- Two levels of table - *Per-process table, System-wide open-file-table.*
    - Open file information – File pointer, File open count, Disk location of the file.
- **Access Methods** – Sequential access, Direct access (relative access).
- **Directory structure**–
    - Advantages – Efficiency, Naming, Grouping.
    - Single-level directory (A single directory for all users).
    - Two-level directory- two-level tree (separate directory for each user).
    - Tree-structure directories (A directory contains a set of files or subdirectories)
        - Path names – Absolute, Relative.
    - Acyclic-graph directories (allows directories to have shared subdirectories and files)
        - File may have several absolute path names.
        - <u>Disadvantage</u> - Dangling pointer problem.
    - General Graph Directory (Allow only links to files (not subdirectories).
        - Physically delete when link count is zero.
        - <u>Disadvantage</u> – If there is self-pointer, will not delete. Require garbage collector.
- File System Mounting - mounted at a **mount point**.
- **Protection** (read, write, execute, append, delete, list)
- Access Lists and Groups (Owner access, Group access, Public access)
- File system is organized into layers *(I/O control, Basic FS, File organization module, Logical FS, Application programs)*
- File Control Block (FCB) - ownership, permissions, location of the file contents, etc.
- Directory Implementation –
    - Linear list (Simple, linear search time-consuming)
    - Hash table (Decreases search time, collisions)
- Allocation Methods (physical) –
    - **Contiguous allocation**
        - <u>Advantages</u> - Simple, Support sequential and direct accesses.
        - <u>Disadvantages</u>- Dynamic storage-allocation problem (due to external fragmentation), Files can't grow.
    - **Linked allocation** -Each block contains a pointer to next block.
        - <u>Advantages</u>- No external fragmentation, Files can grow.
        - <u>Disadvantages</u> - Efficient only for sequential accesses, Reliability issue: if a pointer is lost/damaged.
        - *File-allocation table* (FAT) - a variant of linked allocation.
    - **Indexed allocation** - Each file has its own index block. (an array of disk block addresses)
        - Index block size -
            - Too small, cannot support large files
            - Too large, waste space
        - Indexed Allocation Mapping –
            - Linked scheme
            - Two-level index
- **Free-Space Management** -
    - Bit vector/ bit map (block is represented by 1 bit: free (1), allocated(0)) – Require more space.
    - Linked list (One block point to another block. Not efficient, Cannot get contiguous space easily)
    - Grouping (Store the addresses of *n* free blocks in the first free block. Another in another)
    - Counting (Keep the address of the first free block and the number *n* of free contiguous blocks)
- Efficiency and Performance
    - Performance - *Disk cache, Free- behind & read-ahead, dedicating section of memory as virtual disk / RAM disk.*

- Page Cache - caches pages rather than disk blocks.
    - Unified Buffer Cache(uses the same page cache to cache both memory-mapped pages and ordinary FS I/O)

Lecture 9 – I/O (*Reading Assignment 1*)

- Two main jobs of a computer: I/O and Processing.
- I/O port consists of 4 registers – status, control, Data-in, Data-out.
- A *device controller* is a collection of electronics that can operate a port, a bus, or a device.
- Device addresses – I/O address space, CPU address space. Some use both.
- Polling – to determine the state of a device.(When polling is not efficient, use *interrupt*.)
- Interrupts – Non-maskable(like unrecoverable memory errors), Maskable(can be turned off by CPU when CS execution)
- **Basic interrupt mechanism -**
    1) I/O raises an interrupt to the CPU's interrupt request line.
    2) CPU catches and dispatches to the *interrupt handler*.
    3) Interrupt handler receives, determine cause, perform processing & execute return-from-interrupt instruction.
- **OS interaction with the interrupt mechanism**–At boot time, During I/O.
- **Direct Memory Access (DMA)** bypasses CPU to transfer data directly between I/O device and memory.
    - Handshakingbetween DMA controller and device controller - **DMA-request line** and **DMA-acknowledge line.**
- Characteristics of I/O devices - //slide 16
- Block Devices (read, write, seek) and Character Devices (get, put)
- Network Devices (pipes, FIFOs, streams, queues, mailboxes)
- **Clocks and Timers**– provide current time, provide elapsed time and set timer to trigger operation.
    - **Programmable Interval Timer** is used for timings, and periodic interrupts.
- **Blocking and Non-blocking I/O**
    - Blocking – process suspended until I/O completed.
    - Non-blocking – I/O call returns as much data as available, without waiting for I/O to complete.
    - *Asynchronous* I/O – process runs while I/O executes.
- **Kernel I/O Subsystem**
    - I/O scheduling.
    - Buffering (Buffer is a memory area that stores data being transferred)
        - To cope with device speed mismatch. (modem)
        - To cope with device transfer size mismatch. (network fragmentation and assembly)
        - To maintain *copy semantics.*(version written to disk and version in application must be same)
    - Caching (cache is different from a buffer)
        - Cache is important to improve the access time to the data item that is accessed frequently.
    - Spooling (A spool is a buffer that holds output for a device)
    - Device reservation (provides exclusive access to a device)
    - Error handling.
- **Life Cycle of An I/O Request** - //slide 28
- Improving performance
    - Reduce context switches / frequency of interrupts.
    - Increase buffer size.
    - Concurrency using DMA controllers.
- **Disk Scheduling** - Access time (Seek time + Rotational Latency), Disk bandwidth.
    - **First Come first Served (FCFS)** – Simplest disk scheduling algorithm.
    - **Shortest Seek Time First (SSTF)** – Selects the request with minimum seek time from current head position.
    - **SCAN / elevator algorithm** - Disk arm starts at one end and moves toward the other end, servicing requests. Then the head movement is reversed and servicing continues.
    - **C-SCAN** – Similar to SCAN, but no servicing when reversing. (When end is reached, return arm to beginning)
    - **C-LOOK** – Goes until last request and reverses immediately.
- **Other Disk Management** – Disk formatting, Boot block, Bad blocks, Swap Space Management.

- RAID (*redundant array of independent disks*) – improves speed, improves reliability.
    - Level 0 (performance), Level 1 (mirrored), Level 2, Level 3 (Bit-interleaved parity), Level 4 (Block-level parity), Level 5 (Block-level distributed parity)
- **Disk Attachment**– Host attached or Network attached.

Lecture 10 – Protection& Security (*Reading Assignment 2*)

- Protection problem: to ensure each object is accessed correctly only by processes that are allowed to do so.
    - *Mechanism* (How it will be done)
    - *Policy* (Decide what will be done)
- Protection domain: specifies the resources that a process may access.
    - Can be static or dynamic.
- Domain can be - each user, each process, or each procedure.
    - Can be in monitor mode (privileged) or user mode (non-privileged)
- Protection can be implemented by -
    - **Access matrix**- the *mechanism* for protection.

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read, write		read, write	

        - Can be expanded to the dynamic protection (copy, owner, control)
        - Implementation of Access Matrix
            - Each column = access list for one object.
            - Each row = capability list for one domain.
    - **Global table** – consists of set of ordered triples.<Domain, Object, Rights>
        - <u>Drawbacks</u>–
            - Large table, cannot be in memory, requires disk I/O.
            - Doesn't take advantage of grouping of objects or domains.
    - **Access lists**–One list per object. <Domain, Rights-Set>
    - **Capability lists** - List of objects together with the operations allowed on those objects.
        - Tag associated with each object.
        - Split address space into two parts
            - Part A: Normal address space (instructions, data etc.)
            - Part B: Capability list (only accessible from operating system)
    - **Lock-key mechanism** - Compromise between access list and capability list.
- **The security problem** - considers external environment of the system.
    - Four levels –Physical, Human, OS, Network.
    - **Authentication** -
        - User possessions→ what the user possess (a key or a card).
        - User knowledge → user identifier and password.
        - User attributes→ finger print, retina pattern, or signature.
    - **Program Threats** –
        - Trojan Horse–misuses its environment. Exploits mechanism.
        - Trap Door - designer of a program leaves a hole that only she/he is capable of using.
        - Stack and Buffer Overflow - Exploits a bug in a program.
            - <u>Solution</u> - CPU disallows execution of code in a stack section of memory.
        - Virus - fragment of code that is embedded in a legitimate program.
    - **System Threats** –
        - Denial of Service: Overload the targeted computer preventing it from doing any useful work.
        - Worm.

- Two management techniques
    - Threat monitoring: check for suspicious patterns of activity.
    - Audit log: record the time, the user, and the type of all accesses to an object.
- **Intrusion Detection, Firewall, Encryption.**
- Computer Security Classifications (by US Defense) – A, B, C, D.