**CS3451 INTRODUCTION TO OPERATING SYSTEMS**

**UNIT I INTRODUCTION**

**1. What is a Virtual Machine? List the Advantages of Virtualization. Explain the Creation of a Virtual Machine with Architecture Diagram**

**[16 Marks] – NOV/DEC 2013**

**1. What is a Virtual Machine (VM)?**

A **Virtual Machine** is a **software emulation** of a physical computer. It provides an environment that behaves like an actual computer with its own **CPU, memory, disk, and network interface**, but it runs **on top of another host operating system**.

**Types of Virtual Machines:**

1. **System Virtual Machines:**
   * Virtualize the entire hardware.
   * Examples: VMware, VirtualBox.
2. **Process Virtual Machines:**
   * Provide a platform-independent environment for a single process.
   * Example: Java Virtual Machine (JVM).

**2. Advantages of Virtualization**

| **Advantage** | **Description** |
| --- | --- |
| **Resource Utilization** | Multiple VMs share a single physical system efficiently. |
| **Isolation** | Each VM runs independently without affecting others. |
| **Cost-effective** | Reduces hardware costs by consolidating systems. |
| **Testing & Development** | Ideal for OS/software testing without damaging the host. |
| **Disaster Recovery** | Easy backup and recovery via VM snapshots. |
| **Scalability & Flexibility** | Easy to add/remove VM instances as needed. |
| **Cross-Platform Support** | Run different OS types (e.g., Windows on Linux) on same machine. |

**3. Creation of a Virtual Machine**

A virtual machine is created and managed by a **Virtual Machine Monitor (VMM)**, also called a **Hypervisor**.

**Steps to Create a Virtual Machine:**

1. **Install a Hypervisor** on the host system.
   * Type 1 (Bare-metal): Runs directly on hardware (e.g., Xen, ESXi).
   * Type 2 (Hosted): Runs on host OS (e.g., VirtualBox, VMware Workstation).
2. **Allocate Resources:**
   * Define virtual CPU, memory, disk space, and network settings for the VM.
3. **Install Guest OS:**
   * Load the ISO image and install OS (like Windows, Linux) on the VM.
4. **Start the VM:**
   * The VM runs as an independent system inside the host.

**Architecture Diagram of Virtual Machine System**

pgsql

CopyEdit

---------------------------------------------------

| Application Programs |

---------------------------------------------------

| Guest Operating System |

---------------------------------------------------

| Virtual Machine Monitor (VMM) |

| (Handles resource allocation & management) |

---------------------------------------------------

| Host OS (for Type-2) / Hardware (Type-1) |

---------------------------------------------------

| Physical Hardware (CPU, RAM, etc.) |

---------------------------------------------------

**Explanation of Diagram:**

* Each **Guest OS** thinks it has full control of the hardware.
* **VMM/Hypervisor** intercepts and manages all access to real hardware.
* Multiple VMs can run **different operating systems** on the same hardware simultaneously.

**4. Real-World Examples of Virtualization Tools**

| **Tool/Software** | **Type** | **Use Case** |
| --- | --- | --- |
| VMware Workstation | Type 2 | Desktop virtualization |
| VirtualBox | Type 2 | OS testing, educational purposes |
| VMware ESXi | Type 1 | Data centers, enterprise-level VMs |
| KVM | Type 1 | Linux-based virtualization |
| Hyper-V | Type 1 | Windows-based server virtualization |

**2. (i) State the Basic Functions of OS and DMA**

**(ii) Explain System Calls, System Programs, and OS Generation**  
**[16 Marks Total]**

**(i) Basic Functions of Operating System (OS)**

**[4 Marks]**

An Operating System (OS) is system software that **manages hardware resources** and provides **services to application programs**.

**Basic Functions of OS:**

1. **Process Management:**
   * Manages creation, scheduling, and termination of processes.
   * Allocates CPU time to processes (scheduling).
2. **Memory Management:**
   * Manages allocation and deallocation of memory space.
   * Ensures processes do not interfere with each other’s memory.
3. **File System Management:**
   * Manages data storage, file access, directories, and permissions.
4. **Device Management:**
   * Coordinates and controls input/output (I/O) devices.
   * Uses device drivers and I/O scheduling.
5. **Security and Protection:**
   * Prevents unauthorized access to resources.
   * Provides authentication and access control.
6. **User Interface:**
   * Offers CLI (Command-Line Interface) or GUI (Graphical User Interface).

**Functions of DMA (Direct Memory Access)**

**[4 Marks]**

**DMA** allows **hardware devices** to access main memory **without involving the CPU**, improving performance.

**Functions:**

1. **Transfers Data Directly:**
   * Moves data between memory and devices without CPU intervention.
2. **Reduces CPU Overhead:**
   * Frees up CPU to perform other tasks during data transfer.
3. **Handles Large Data Blocks:**
   * Ideal for fast I/O devices (disk, network).
4. **Improves System Efficiency:**
   * Supports high-speed data transfers.

**(ii) System Calls, System Programs, and OS Generation**

**[8 Marks]**

**1. System Calls:**

System calls provide an **interface between the user programs and the OS kernel**.

**Types of System Calls:**

| **Type** | **Example Functions** |
| --- | --- |
| **Process Control** | fork(), exit(), wait() |
| **File Management** | open(), read(), write() |
| **Device Management** | ioctl(), read(), write() |
| **Information Maintenance** | getpid(), alarm() |
| **Communication** | pipe(), shmget(), send() |

**2. System Programs:**

System programs provide a **convenient environment** for program development and execution.

**Examples:**

* File manipulation tools (cp, mv, rm)
* Status information (ps, top)
* Programming tools (compilers, debuggers)
* Communication tools (ssh, mail)
* System utilities (disk formatters, backup)

**3. Operating System Generation:**

**OS Generation** refers to the process of **customizing the operating system** for a specific computer system.

**Steps:**

1. **System Configuration:**
   * Determine the hardware and software environment.
2. **OS Modules Selection:**
   * Choose needed OS components (scheduler, memory manager, etc.).
3. **Linking and Compilation:**
   * OS components are compiled and linked to create the final OS.
4. **Installation:**
   * The generated OS is loaded onto the system's disk.

**Tools Used:**

* Configuration scripts
* System generation tools (e.g., SYSGEN in older systems)

**UNIT II PROCESS MANAGEMENT**

**1) Write in detail about Deadlock Avoidance**

**[U] (NOV/DEC 2009)**

**Deadlock:**

Deadlock is a situation where a set of processes are **waiting for each other indefinitely** for resources, causing the system to freeze.

**Deadlock Avoidance:**

Deadlock avoidance ensures the system **never enters** an unsafe state that could lead to deadlock. It requires the **operating system to have prior knowledge** of resource requirements.

**Basic Idea:**

* When a process requests resources, the OS:
  + Checks if the allocation keeps the system in a **safe state**.
  + If yes, the resources are allocated.
  + If not, the process must wait.

**Safe State:**

A state where **there is a sequence of processes** {P1, P2, ..., Pn} such that each process can complete even if all others wait.

**Key Conditions Avoided:**

* Circular wait
* Hold and wait

**Deadlock Avoidance Strategies:**

| **Method** | **Description** |
| --- | --- |
| **Resource Allocation Graph (RAG)** | Used when there is **only one instance per resource**. Avoid cycles. |
| **Banker’s Algorithm** | Used when there are **multiple instances** of resources. Works like a bank giving out loans carefully. |

**Resource Allocation Graph (RAG) with Claim Edges:**

* Add **claim edges** (dotted lines) to represent possible future resource requests.
* Before granting a request, check if adding the request creates a **cycle**.
* If cycle forms → **Reject/Delay request**.

**Advantages:**

* System always stays in a safe state.
* Deadlock is **prevented before it happens**.

**Disadvantages:**

* Requires **advance knowledge** of all resource needs.
* More **complex and resource-intensive**.

**2) Write in detail about Deadlock Recovery**

**[U] (APRIL/MAY 2011)**

**Deadlock Recovery:**

If a system **does not avoid or prevent deadlock**, it must **detect and recover** from it after it occurs.

**Steps in Deadlock Recovery:**

**1. Deadlock Detection:**

* The OS detects deadlock using resource allocation data structures.
* Detection methods differ for **single-instance** or **multiple-instance** resources.

**2. Recovery Techniques:**

**A. Process Termination**

* **Kill All Deadlocked Processes**
  + Simple but loses all work done.
* **Terminate One-by-One**
  + Kill one process at a time until deadlock breaks.
  + Criteria for choosing process:
    - Priority
    - CPU time used
    - Resources held

**B. Resource Preemption**

* Take a resource away from a process and give to another.
* Rollback preempted process to a safe state.
* May cause starvation if same process is chosen repeatedly.

**Disadvantages:**

* Data loss or corruption if processes are forcibly terminated.
* Complex to rollback process states.

**3) Explain the Banker’s Algorithm for Deadlock Avoidance with Example**

**[Ap]**

**Banker’s Algorithm:**

Introduced by **Edsger Dijkstra**, this algorithm simulates a bank allocating resources to customers **without exceeding the total available**.

It is applicable when:

* **Multiple instances** of each resource exist.
* Each process must declare its **maximum resource need** in advance.

**Data Structures:**

| **Structure** | **Description** |
| --- | --- |
| **Available** | Number of available instances of each resource. |
| **Max** | Maximum demand of each process. |
| **Allocation** | Current allocation for each process. |
| **Need** | Remaining resource needs (Max - Allocation). |

**Algorithm Steps:**

1. **Check if the request is less than or equal to Need.**
2. **Check if the request is less than or equal to Available.**
3. Temporarily allocate resources.
4. Run **Safety Algorithm** to check if system remains in a safe state.
5. If safe → Grant request. Else → Rollback allocation.

**Example:**

Assume 3 resource types (A, B, C), and 5 processes (P0 to P4):

**Given:**

yaml

CopyEdit

Available = [3, 3, 2]

Max Matrix:

P0: [7, 5, 3] P1: [3, 2, 2] P2: [9, 0, 2] P3: [2, 2, 2] P4: [4, 3, 3]

Allocation Matrix:

P0: [0, 1, 0] P1: [2, 0, 0] P2: [3, 0, 2] P3: [2, 1, 1] P4: [0, 0, 2]

Need = Max - Allocation

Run **Safety Algorithm**:

* Try to find a process whose Need ≤ Available.
* Simulate its completion → Release its resources → Repeat.
* If all processes can finish, the state is **safe**.

**Advantages:**

* Prevents deadlock completely.
* Ensures safe resource allocation.

**Disadvantages:**

* Requires **prior knowledge** of max resource needs.
* Not suitable for systems with a large number of processes/resources.

**UNIT III MEMORY MANAGEMENT**

**Explain the Segmentation with Paging Implemented in OS/2 32-bit IBM System**

**Also Describe the Following Algorithms:** a. First Fit  
b.BestFit  
c.WorstFit  
**[An] (APRIL/MAY 2010)**

**Segmentation with Paging in OS/2 (32-bit IBM System)**

**[8 Marks]**

**✅ What is Segmentation with Paging?**

* It's a **memory management scheme** that **combines segmentation and paging**.
* Used in IBM's **OS/2 32-bit system** to **reduce fragmentation** and provide **efficient memory protection**.

**Why Combine Segmentation and Paging?**

* **Segmentation**: Divides program into **logical units** (code, data, stack).
* **Paging**: Each segment is **further divided** into fixed-size pages to eliminate **external fragmentation**.

**Architecture of OS/2 32-bit System:**

**1. Logical Address Format:**

java

CopyEdit

Logical Address = <Segment Selector, Offset>

* **Segment Selector**: Indexes into **Segment Descriptor Table**.
* **Offset**: Within the segment, further divided into **page number + page offset**.

**Steps in Address Translation:**

1. Segment Selector accesses the **Segment Descriptor**.
2. Descriptor provides the **base address** of the segment.
3. Offset is split into:
   * **Page number** → selects page from **Page Table**.
   * **Page offset** → locates byte within the page.
4. Final **physical address** = base + page table lookup + offset.

**Benefits:**

* **Segmentation** provides logical separation and protection.
* **Paging** avoids fragmentation and makes memory allocation flexible.
* Enables **virtual memory** in OS/2 32-bit systems.

**Diagram: Segmentation with Paging**

pgsql

CopyEdit

+---------------------------+

| Segment Selector (16-bit)|

+---------------------------+

|

v

+----------------------------+

| Segment Descriptor Table |

+----------------------------+

|

v

+----------------------------+

| Segment Base Address |

+----------------------------+

|

v

+-----------------------------+

| Page Table for the Segment |

+-----------------------------+

|

v

+----------------------+

| Physical Frame + Offset |

+----------------------+

**a. First Fit Algorithm**

**[2 Marks]**

**Definition:**

* Allocates the **first block** of memory that is **large enough** for the process.

**Working:**

* Scans memory from the beginning.
* As soon as a **suitable hole** is found, it is allocated.

**Example:**

Free memory: 100 KB, 500 KB, 200 KB  
Request: 150 KB → Allocated in **500 KB**

**✅ Advantage: Fast, simple to implement.**

**❌ Disadvantage: May cause small unusable holes (fragmentation).**

**b. Best Fit Algorithm**

**[2 Marks]**

**Definition:**

* Allocates the **smallest hole** that is **big enough** for the request.

**Working:**

* Scans entire memory, selects the **best-fitting block** (minimal waste).

**Example:**

Free memory: 100 KB, 500 KB, 200 KB  
Request: 150 KB → Allocated in **200 KB**

**✅ Advantage: Minimizes wasted space.**

**❌ Disadvantage: Slower (must check all blocks), causes fragmentation.**

**c. Worst Fit Algorithm**

**[2 Marks]**

**Definition:**

* Allocates the **largest available** hole.

**Working:**

* Goal: Leave **larger remaining holes** for future allocation.

**Example:**

Free memory: 100 KB, 500 KB, 200 KB  
Request: 150 KB → Allocated in **500 KB**

**✅ Advantage: May reduce small fragments.**

**❌ Disadvantage: Can waste large chunks of memory.**

**✅ Final Summary Table:**

| **Algorithm** | **Strategy** | **Pros** | **Cons** |
| --- | --- | --- | --- |
| First Fit | First available hole | Fast | Causes fragmentation |
| Best Fit | Smallest suitable hole | Saves space | Slower, fragmented |
| Worst Fit | Largest hole | Reduces small leftover holes | Can waste large memory |

**2. (i) Compare Paging with Segmentation in Terms of Memory Requirement for Address Translation**

**(ii) Explain Page Replacement Algorithms with Suitable Example**

**[An] (NOV/DEC 2021)**

**(i) Paging vs Segmentation – Memory Requirement for Address Translation**

**[8 Marks]**

| **Feature** | **Paging** | **Segmentation** |
| --- | --- | --- |
| **Basic Concept** | Divides memory into fixed-size blocks called **pages** | Divides memory into **variable-size segments** |
| **Address Structure** | Virtual address = Page number + Offset | Virtual address = Segment number + Offset |
| **Address Translation** | Uses **page table** to map virtual to physical pages | Uses **segment table** with base and limit |
| **Translation Data Size** | Requires a **page table** for each process | Requires a **segment table** for each process |
| **Memory Requirement** | Needs memory for storing page table entries for all pages | Needs fewer entries (as segments are fewer) |
| **Fragmentation** | Internal fragmentation possible | External fragmentation possible |
| **Access Time Overhead** | Higher due to more table lookups | Less overhead in comparison |
| **Example** | Page Table: 1024 pages = 1024 entries | Segment Table: 5-10 entries |

**Conclusion**:  
Paging requires **more memory** for address translation structures due to larger page tables, while segmentation requires **less memory**, since segments are fewer and larger in size.

**(ii) Page Replacement Algorithms**

**[8 Marks]**

When a page fault occurs and memory is full, the **page replacement algorithm** decides which page to remove to make space.

**🔹 1. FIFO (First In First Out)**

* Removes the **oldest page** in memory.

**Example:**

Reference String: 7 0 1 2 0 3 0 4 (3 frames)  
Replacements: 7 0 1 (page fault) → 2 (removes 7) → 3 (removes 0) etc.

✅ Simple  
❌ May replace heavily used pages

**🔹 2. LRU (Least Recently Used)**

* Removes the page that **hasn’t been used for the longest time**.

**Example:**

Reference String: 1 2 3 4 1 2 5 1 2 3 4 5 (3 frames)

* LRU replaces least recently used page each time.

✅ More efficient than FIFO  
❌ Requires tracking usage (complex)

**🔹 3. Optimal Replacement**

* Replaces the page that **won’t be used for the longest time in future**.

**Example:**

Reference String: 7 0 1 2 0 3 0 4

* When 2 comes in and memory is full: look ahead, choose the page that won't be used soonest.

✅ Minimum page faults  
❌ Cannot be implemented in practice (needs future knowledge)

**🔹 4. Clock (Second Chance)**

* Uses a circular list (like a clock).
* If a page's reference bit = 0, it's replaced.
* If 1, give a second chance (bit set to 0).

✅ Efficient approximation of LRU  
❌ Still not as optimal as actual LRU

**✅ Comparison Table:**

| **Algorithm** | **Simple** | **Efficient** | **Realistic** | **Page Faults** |
| --- | --- | --- | --- | --- |
| FIFO | ✅ | ❌ | ✅ | Medium |
| LRU | ❌ | ✅ | ✅ (with support) | Low |
| Optimal | ❌ | ✅✅ | ❌ | Minimum |
| Clock | ✅ | ✅ | ✅ | Moderate |

**Final Notes:**

* Efficient page replacement is critical to system performance.
* Real-world systems use **approximations of LRU**, like **Clock** algorithm.

**UNIT IV STORAGE MANAGEMENT**

**1) Explain and Compare FCFS, SSTF, C-SCAN and C-LOOK Disk Scheduling Algorithms with Examples**

**[An] (NOV/DEC 2012)**

Disk scheduling algorithms determine the **order in which disk I/O requests** are serviced to improve **performance** and **reduce seek time**.

Assume:

* Disk Queue: 98, 183, 37, 122, 14, 124, 65, 67
* Initial head position: 53

**🔹 1. FCFS (First-Come-First-Serve)**

**Mechanism**:

* Services requests **in the order they arrive**.

**Example Seek Order**:  
53 → 98 → 183 → 37 → 122 → 14 → 124 → 65 → 67

**Total Head Movement** =  
|98-53| + |183-98| + |37-183| + ... = **640 cylinders**

✅ **Advantage**: Simple  
❌ **Disadvantage**: Poor performance, high seek time

**🔹 2. SSTF (Shortest Seek Time First)**

**Mechanism**:

* Services the request **closest to current head position**.

**Seek Order** (starting at 53):  
53 → 65 → 67 → 37 → 14 → 98 → 122 → 124 → 183

**Total Head Movement** =  
12 + 2 + 30 + 23 + 84 + 24 + 2 + 59 = **236 cylinders**

✅ **Advantage**: Reduces seek time  
❌ **Disadvantage**: Starvation of distant requests

**🔹 3. C-SCAN (Circular SCAN)**

**Mechanism**:

* Services requests in **one direction** only (e.g., towards higher cylinders),
* Then jumps to the **beginning** and continues.

**Seek Order** (start at 53):  
53 → 65 → 67 → 98 → 122 → 124 → 183 → [jump to 0] → 14 → 37

**Total Head Movement** =  
12 + 2 + 31 + 24 + 2 + 59 + 183 + 14 + 23 = **350 cylinders**

✅ **Advantage**: Uniform wait time  
❌ **Disadvantage**: More total movement due to reset

**🔹 4. C-LOOK (Circular LOOK)**

**Mechanism**:

* Similar to C-SCAN, but **only goes as far as last request**, then jumps to **lowest pending request**.

**Seek Order**:  
53 → 65 → 67 → 98 → 122 → 124 → 183 → [jump to 14] → 37

**Total Head Movement** =  
12 + 2 + 31 + 24 + 2 + 59 + (183-14) + 23 = **322 cylinders**

✅ **Advantage**: Better than C-SCAN, avoids unnecessary scan  
❌ **Disadvantage**: Still has overhead of jump

**✅ Comparison Table**

| **Algorithm** | **Strategy** | **Starvation** | **Seek Time** | **Head Movement** | **Realistic Use** |
| --- | --- | --- | --- | --- | --- |
| FCFS | First-come requests | No | High | High | Basic systems |
| SSTF | Closest request next | Yes | Low | Low | Good for light loads |
| C-SCAN | One-direction then jump | No | Moderate | Moderate | Multi-user OS |
| C-LOOK | One-direction (actual range) | No | Lower | Lower | Improved version of C-SCAN |

**🔚 Conclusion:**

* **FCFS** is fair but inefficient.
* **SSTF** is efficient but can starve.
* **C-SCAN** provides uniform response time.
* **C-LOOK** optimizes C-SCAN by avoiding unnecessary movement.

**2. (i) Discuss the Functions of Files and File Implementation**

**(ii) Explain Free Space Management with Neat Example**

**[U] (NOV/DEC 2015)**

**🔸 (i) Functions of Files and File Implementation**

**[8 Marks]**

**✅ Functions of Files in an Operating System:**

1. **Data Storage**
   * Files store both user and system data permanently on secondary storage.
2. **Access Control**
   * OS provides access control to ensure that only authorized users can access specific files.
3. **File Organization**
   * Files can be organized in directories, making it easier to retrieve and manage them.
4. **Data Sharing**
   * Files can be shared among multiple users or processes.
5. **Security and Protection**
   * Files are protected using permissions (read/write/execute).
6. **Concurrency Control**
   * Multiple processes accessing files simultaneously are managed to prevent inconsistency.

**✅ File Implementation Methods:**

File implementation refers to **how files are stored** on the disk. There are 3 major methods:

1. **Contiguous Allocation**
   * Files occupy a set of contiguous blocks on disk.
   * ✅ Simple and fast
   * ❌ May suffer from external fragmentation.
2. **Linked Allocation**
   * Each file is a linked list of disk blocks.
   * ✅ No external fragmentation
   * ❌ Slower access for random data
3. **Indexed Allocation**
   * Each file has an index block with pointers to data blocks.
   * ✅ Direct access possible
   * ❌ Overhead of index block

**🔸 (ii) Free Space Management**

**[8 Marks]**

Free space management is the method of **tracking unallocated disk blocks** so that they can be used efficiently when needed.

**✅ Techniques for Free Space Management:**

1. **Bit Vector (Bitmap)**
   * Each block represented by a bit: 1 if allocated, 0 if free.

**Example** (for 8 blocks):  
1 0 0 1 1 0 0 0  
→ Free blocks: 1, 2, 5, 6, 7

✅ Easy to find free blocks  
❌ Requires extra memory

1. **Linked List**
   * Free blocks linked together (each block points to the next free block).

**Example**:  
Block 3 → Block 6 → Block 9 → NULL

✅ Simple  
❌ Must traverse list to find free space

1. **Grouping**
   * Stores addresses of free blocks in groups.

Example:  
First block contains addresses of next 99 free blocks. The 100th block contains addresses of the next 100 free blocks, and so on.

✅ Speeds up allocation  
❌ More complex

1. **Counting**
   * Tracks beginning address and number of contiguous free blocks.

Example:  
(Start = 5, Count = 3) → Blocks 5, 6, 7 are free

✅ Efficient for large free areas  
❌ Less flexible for scattered free space

**✅ Conclusion:**

* **File systems** provide efficient storage and retrieval mechanisms.
* **Free space management** is essential to keep track of available storage and ensure optimal performance.

**UNIT V VIRTUAL MACHINES AND MOBILE OS**

**1. (i) Explain the Components of Linux System with Neat Sketch (6 Marks)**

**✅ Components of Linux System:**

Linux is a **modular and layered architecture**, composed of several interacting parts:

**🔹1. Kernel**

* Core part of the Linux OS.
* Manages hardware, memory, processes, filesystems, and device drivers.

**🔹2. System Libraries**

* Standard libraries that **interact with the kernel**.
* Example: GNU C Library (glibc)

**🔹3. System Utilities**

* Essential tools for file manipulation, process control, user management.
* Example: cp, mv, top, ps

**🔹4. Shell**

* Command-line interpreter.
* Converts user commands into kernel instructions.
* Example: Bash, Zsh

**🔹5. Application Programs**

* User-level programs such as editors (vim, nano), browsers, etc.

**🔹 Neat Sketch of Linux Architecture:**

sql

CopyEdit

+--------------------------+

| Application Layer |

+--------------------------+

| Shell |

+--------------------------+

| System Utilities |

+--------------------------+

| System Libraries |

+--------------------------+

| Kernel |

| - Process Mgmt |

| - Memory Mgmt |

| - Device Drivers |

+--------------------------+

| Hardware (CPU, RAM) |

+--------------------------+

**(ii) Various System Administrator Roles in LINUX OS (7 Marks)**

A **Linux system administrator** ensures that Linux systems run smoothly and securely.

**✅ Roles and Responsibilities:**

1. **User Management**
   * Add/delete/modify users and groups.
   * Set permissions and ownership.
2. **Software Installation & Updates**
   * Install packages using yum, apt, dnf.
   * Maintain updated system software.
3. **Disk & File System Management**
   * Partitioning, formatting, and mounting drives.
   * Manage file systems like ext4, xfs, etc.
4. **System Monitoring**
   * Monitor performance using tools like top, htop, vmstat.
5. **Backup & Recovery**
   * Schedule backups using tools like rsync, tar, cron.
   * Plan recovery strategies.
6. **Security & Firewall Management**
   * Configure iptables, ufw, and SELinux.
   * Apply updates to patch vulnerabilities.
7. **Service & Process Management**
   * Start/stop services using systemctl.
   * Manage background processes (ps, kill, nice).
8. **Network Configuration**
   * Set IP addresses, DNS, routing.
   * Manage network services (see question 19 below).

**2. (i) How to Install and Configure Network Services in LINUX (9 Marks)**

Installing and configuring network services allows a Linux system to function in a networked environment.

**✅ Steps for Installing and Configuring Network Services:**

**🔹1. Install Required Network Packages**

* Use package manager:

bash

CopyEdit

sudo apt install net-tools openssh-server

sudo yum install httpd vsftpd

**🔹2. Assign IP Address & Hostname**

* Temporary (ifconfig):

bash

CopyEdit

sudo ifconfig eth0 192.168.1.10 netmask 255.255.255.0

* Persistent (for Ubuntu): Edit /etc/netplan/\*.yaml  
  For CentOS: /etc/sysconfig/network-scripts/ifcfg-eth0

**🔹3. Enable and Start Network Services**

* Example: Start SSH

bash

CopyEdit

sudo systemctl start sshd

sudo systemctl enable sshd

* Example: Start Apache

bash

CopyEdit

sudo systemctl start httpd

sudo systemctl enable httpd

**🔹4. Open Ports in Firewall**

bash

CopyEdit

sudo firewall-cmd --permanent --add-service=http

sudo firewall-cmd --reload

**🔹5. Verify Services**

* Use netstat -tulnp or ss -tuln
* Use browser or telnet to verify response

**🔹6. Test Remote Access**

bash

CopyEdit

ssh user@server\_ip

**✅ Common Network Services:**

| **Service** | **Purpose** | **Default Port** |
| --- | --- | --- |
| SSH | Remote access | 22 |
| HTTP | Web server | 80 |
| FTP | File transfer | 21 |
| DHCP | Auto IP assignment | 67/68 |
| DNS | Domain resolution | 53 |

**3. Deadlock and Resource Allocation Graph Problem**

**[An] (APR/MAY 2017)**

**🔹 1. What is a Deadlock?**

A **deadlock** is a situation in which a set of processes are **blocked indefinitely** because each process is holding a resource and waiting to acquire a resource held by another process.

**🔹 2. Necessary Conditions for Deadlock**

Deadlock can occur **only if all the following four conditions hold simultaneously:**

1. **Mutual Exclusion**
   * At least one resource must be held in a non-shareable mode.
2. **Hold and Wait**
   * A process is holding at least one resource and waiting to acquire more.
3. **No Preemption**
   * Resources cannot be forcibly removed from a process.
4. **Circular Wait**
   * A circular chain of two or more processes exists, where each process is waiting for a resource held by the next process in the chain.

**🔹 3. Deadlock Prevention**

Deadlock prevention aims to **break at least one of the four necessary conditions**:

| **Condition** | **Prevention Strategy** |
| --- | --- |
| Mutual Exclusion | Not possible for some resources like printers |
| Hold and Wait | Require a process to request all resources at once |
| No Preemption | If a process is waiting, forcibly preempt and reassign its resources |
| Circular Wait | Impose a fixed ordering of resource allocation |

**🔸 Problem Setup: Resource Allocation Graph**

**➤ Resources:**

* 2 instances of R1
* 2 instances of R2

**➤ Processes:**

* P1 to P4

**➤ Current Allocation:**

* R1: 1 to P2, 1 to P3
* R2: 1 to P1, 1 to P4

**➤ Requests:**

* P1 → wants 1 R1
* P3 → wants 1 R2

**🔹 1. Resource Allocation Graph**

plaintext

CopyEdit

P1 → R1 [P1 holds R2, wants R1]

P2 ← R1 [P2 holds R1]

P3 → R2 [P3 holds R1, wants R2]

P4 ← R2 [P4 holds R2]

**Graph Elements:**

* **Edges from process → resource** = Request
* **Edges from resource → process** = Allocation

**🔹 2. State of Each Process**

| **Process** | **Holding** | **Requesting** | **State** | **Waiting For** |
| --- | --- | --- | --- | --- |
| P1 | 1 instance of R2 | 1 R1 | Waiting | R1 |
| P2 | 1 instance of R1 | – | Runnable | – |
| P3 | 1 instance of R1 | 1 R2 | Waiting | R2 |
| P4 | 1 instance of R2 | – | Runnable | – |

**🔹 3. Is the System Deadlocked?**

Let’s **simulate** possible execution:

**Step 1:**

* **P2** is runnable, holds R1, does not need anything — **completes and releases R1**

**Step 2:**

* **P1** was waiting for R1 — now available — **P1 runs**, uses R1 and R2, then **completes and releases both**

**Step 3:**

* Now **P3** was waiting for R2 — R2 is available — **P3 runs**, uses R1 and R2, then **completes**

**Step 4:**

* **P4** is already holding R2 and doesn’t need anything — **completes**

✅ **All processes finish**. So, **NO DEADLOCK**

**✅ Execution Sequence:**

1. P2 → release R1
2. P1 → acquires R1 → executes → releases R1 and R2
3. P3 → acquires R2 → executes → releases R1 and R2
4. P4 → completes (already had R2)

**✅ Final Answer Summary:**

* Deadlock: **No**
* State:
  + P1 & P3 → Waiting initially
  + P2 & P4 → Runnable
* Execution Order: **P2 → P1 → P3 → P4**