Exercise 1

Definition:

- An operating system is an interface between the hardware of a computer and the user (programs or humans).
- An operating system is a program (or a set of programs) that facilitates the execution of other programs.
- An operating system acts as a general manager supervising the activity of each component in the computer system.

Two major design goals of an operating system are:

- Efficient use of hardware.
- Ease of use of resources.

Exercise 2

Bootstrap Process

- **Power On:** The process begins when the computer is powered on or restarted.
- **POST (Power-On Self-Test):** The BIOS/UEFI firmware performs a POST to check the hardware components and ensure everything is functioning correctly.
- **BIOS/UEFI Initialization:** The BIOS/UEFI firmware initializes the system hardware and configures settings such as the clock and memory.
- **Bootstrap Program (Bootloader) Activation:** The BIOS/UEFI locates and executes the bootstrap program from a predefined storage device (HDD, SSD, USB, etc.).
- **Bootloader Execution:** The bootloader (such as GRUB, LILO, or Windows Boot Manager) is loaded into memory and executed. Its primary role is to load the operating system kernel.
- **Kernel Loading:** The bootloader locates the operating system kernel, loads it into memory, and hands over control to it.
- **Kernel Initialization:** The operating system kernel initializes the rest of the hardware, sets up memory management, and starts system processes.

- **System Services Start:** Essential system services and background processes are started.
- **User Interface Launch:** The graphical user interface (GUI) or command-line interface (CLI) is launched, allowing user interaction.

Role of the Bootstrap Program:

- **Locate and Load OS:** The primary role of the bootstrap program is to locate the operating system kernel on the storage device and load it into memory.
- **Initial Setup:** It sets up the necessary parameters and environment required for the OS to start.
- **Transfer Control:** After loading the kernel, it transfers control to the operating system to continue the boot process.

Exercise 3

Evolution of Operating Systems

- **Batch Systems**
- **Description:** Early computers used batch processing where jobs (tasks) were collected and processed in groups (batches) without user interaction.
- **Key Features:** No direct interaction with the user; jobs were executed sequentially; efficient for large, repetitive tasks.
- **Time-Sharing Systems**
- **Description:** Developed to allow multiple users to interact with the computer simultaneously by sharing CPU time.
- **Key Features:** Multitasking; users interact with the system through terminals; improved resource utilization and user experience.
- **Personal Systems**
- **Description:** The emergence of personal computers (PCs) brought operating systems designed for individual use.
- **Key Features:** User-friendly interfaces (GUIs); support for a wide range of applications; designed for single users but capable of multitasking.
- **Parallel Systems**

- **Description:** Systems with multiple processors working simultaneously to perform tasks faster and more efficiently.
- **Key Features:** Increased computational power; shared memory architecture; used in scientific computing and high-performance applications.

- **Distributed Systems**
- **Description:** Systems where multiple independent computers communicate and collaborate to complete tasks.
- **Key Features:** Resource sharing across a network; increased reliability and scalability; examples include cloud computing and networked services.

- **Real-Time Systems**
- **Description:** Systems designed to process data and respond to inputs within a guaranteed time frame.
- **Key Features:** Deterministic and predictable response times; used in critical applications such as industrial control, medical devices, and embedded systems.

Exercise 4

- **Single-Programming Memory Management:**
- **Single Task:** Only one program is loaded into memory and executed at a time.
- **Simple Allocation:** Memory allocation is straightforward since there's no need to manage multiple programs.
- **Fixed Partitioning:** Entire memory space is allocated to the single program, eliminating the need for complex partitioning.
- **No Overhead:** Minimal overhead in memory management because there's no context switching or memory protection needed.
- **Limited Utilization:** Inefficient use of memory resources as large portions of memory may remain unused.

- **Multi-Programming Memory Management:**

- **Multiple Tasks:** Multiple programs are loaded and executed simultaneously, sharing the memory.
- **Dynamic Allocation:** Memory must be dynamically allocated and managed among various programs.
- **Partitioning:** Memory is divided into partitions, which can be fixed or variable in size, to accommodate multiple programs.
- **Overhead:** Increased overhead due to the need for managing memory allocation, context switching, and memory protection.
- **Efficient Utilization:** Better utilization of memory resources as multiple programs can run concurrently, filling memory gaps and improving system throughput.

Exercise 5

- **Concept of Virtual Memory:**
- Abstraction Layer: Virtual memory provides an abstraction layer between the physical memory (RAM) and the programs running on the system.
- Illusion of Infinite Memory: It gives the illusion of a much larger memory space than is physically available by using disk storage as an extension of RAM.
- Memory Address Translation: Virtual memory allows programs to use memory addresses that are not directly mapped to physical RAM.
- **Functionality in Modern Operating Systems:**
- **Demand Paging:** Only portions of a program that are actively used are loaded into physical memory from disk, reducing initial loading time.
- **Page Replacement:** When physical memory becomes full, the operating system swaps out less frequently used pages to disk and brings in needed pages, utilizing a page replacement algorithm (e.g., LRU, FIFO).
- **Page Table:** Each process has its page table, which maps virtual addresses to physical addresses, facilitating memory address translation.
- **Memory Protection:** Virtual memory systems provide memory protection by assigning access permissions to different memory regions, preventing unauthorized access, and ensuring system stability.

- **Memory-Mapped Files:** Virtual memory allows files to be mapped directly into memory, enabling efficient file I/O operations without explicit reading and writing.
- **Improves System Stability and Efficiency:** Virtual memory enables efficient utilization of physical memory resources, improves system stability by preventing memory fragmentation, and allows for smoother multitasking.

Exercise 6

- **Definition of a Process:**
- **Execution Instance:** A process is an instance of a program that is currently executing on the system.
- **Independent Entity:** It represents an independent unit of work with its own memory space, resources, and execution state.
- **Process Management:**
- **Creation and Termination:** Operating systems create and terminate processes based on user requests or system events.
- **Process Scheduling:** Determines which process gets access to the CPU and for how long, optimizing resource utilization and system responsiveness.
- **Context Switching:** Operating systems perform context switches to save and restore the execution state of processes, allowing for multitasking.
- **Inter-Process Communication (IPC):** Facilitates communication and data exchange between processes through mechanisms like pipes, shared memory, and message passing.
- **Synchronization:** Ensures orderly execution and prevents race conditions among processes by using synchronization primitives like locks, semaphores, and monitors.
- **Deadlock Detection and Handling:** Operating systems detect and resolve deadlocks, where processes are unable to proceed due to resource contention, using techniques such as deadlock detection algorithms and resource allocation strategies.
- **Resource Management:** Manages system resources (CPU time, memory, I/O devices) among processes, ensuring fair and efficient allocation.

- **Process States:** Processes transition through various states (e.g., running, ready, blocked) during execution, managed by the operating system's scheduler and kernel.

Exercise 7

Problems:

- **Race Conditions:** Concurrent access to shared resources by multiple processes may lead to unpredictable outcomes due to timing variations.
- **Deadlocks:** Processes may end up waiting indefinitely for resources held by other processes, resulting in a deadlock where none can proceed.
- **Starvation:** Some processes may be deprived of accessing critical resources for extended periods due to unfair scheduling.

Solutions:

- **Mutual Exclusion (Mutex):**
- **Problem:** Ensuring only one process accesses a shared resource at a time.
- **Solution:** Use mutex locks to allow only one process to hold the lock at a time, ensuring mutual exclusion.
- **Semaphores:**
- **Problem:** Controlling access to a finite number of resources among multiple processes.
- **Solution:** Use semaphores, which are counters used to control access to resources. Semaphores can be binary (mutex) or general (counting) semaphores.
- **Monitors:**
- **Problem:** Managing synchronization and data access in a structured and modular way.
- **Solution:** Use monitors, and high-level synchronization constructs that encapsulate shared data and synchronization operations within a single module.
- **Deadlock Prevention and Avoidance:**
- **Problem:** Preventing or avoiding situations where processes are unable to proceed due to resource contention.

- **Solution:** Implement strategies such as resource allocation graphs, deadlock detection algorithms, and resource allocation policies to prevent or avoid deadlocks.

- **Priority Inversion Prevention:**
- **Problem:** Ensuring high-priority processes do not get blocked by lower-priority processes holding shared resources.
- **Solution:** Use priority inheritance or priority ceiling protocols to temporarily boost the priority of processes holding resources required by higher-priority processes.

- **Synchronization Primitives:**
- **Problem:** Providing low-level mechanisms for coordinating access to shared resources.
- **Solution:** Use synchronization primitives such as atomic operations, test-and-set, wait, and signal to implement higher-level synchronization constructs like locks, semaphores, and monitors.

Exercise 8

- **Functions of Device Manager:**
- **Driver Management:** Installs, configures, and updates device drivers which act like translators between the operating system and the hardware devices
- **Device Identification:** Provides information about the devices connected to your system, like type, manufacturer, and current status.
- **Resource Management:** Allocates resources like interrupt requests (IRQs) and DMA channels to devices to avoid conflicts.
- **Device Monitoring:** Keeps an eye on the health of the devices and reports any errors that might arise.
- **Plug and Play:** Supports the dynamic addition and removal of devices, making it easier to connect and disconnect peripherals.

- **I/O Devices Management:**
- **Device Drivers:** As mentioned earlier, device drivers are crucial for communication between the OS and the device. The device manager loads the appropriate driver for each I/O device.

- **Interaction with Drivers:** Once loaded, the device driver provides specific routines (set of instructions) that the operating system uses to interact with the I/O device.
- **Resource Allocation:** The device manager plays a vital role in managing conflicts between devices by allocating resources like IRQs and DMA channels to prevent them from interrupting each other's operations.

Exercise 9

Functions

- **File Organization:** Create, delete, rename, and move files and folders.
- **File Viewing:** Open and display the contents of files.
- **File Searching:** Locate specific files based on name, type, or content.
- **File Management:** Copy, cut, and paste files and folders.
- **Permissions Management:** Control access rights for users and groups.

Exercise 10

Characteristic	UNIX	Linux	Windows
Kernel Type	Monolithic	Monolithic	Hybrid
Source Model	Pro	Open Source	Closed source
User Inteface	CLI (Command Line Interface)	CLI and GUI (Graphical User Interface)	GUI (Graphical User Interface)
File System	Various (e.g., UFS, ZFS)	Various (e.g., ext4, XFS)	NTFS, FAT32, exFAT
Multitasking	Yes	Yes	Yes
Process Management	Process control blocks (PCBs)	Process control blocks (PCBs)	Process control blocks (PCBs)
Memory Management	Virtual memory with paging	Virtual memory with paging	Virtual memory with paging
Networking	Built-in networking capabilities	Built-in networking capabilities	Built-in networking capabilities

Security	Strong focus on security and permissions	Strong focus on security and permissions	Strong focus on security and permissions
Package Management	Package managers (e.g., apt, yum)	Package managers (e.g., apt, yum)	MSI (Microsoft Installer), Package Manager
File Permission	UNIX- style permissions (rwx)	UNIX- style permissions (rwx)	ACLs (Access Control Lists)
Customizability	Highly customizable, with numerous distributions	Highly customizable, with numerous distributions	Highly customizable, with numerous distributions
Cost	Varies (some versions are free, others require licensing fees)	Free (with some commercial distributions)	Requires licensing fees for most versions
Usage	Commonly used in servers, supercomputers, and embedded systems	Widely used in servers, desktop, smartphones, and embedded systems	Dominant in desktop PCs and widely used in servers
Example	FreeBSD, macOS (based on UNIX), Solaris	Ubuntu, CentOS, Fedora, Debian	Windows 10, Window Server, Window Embedded