GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Plot No.2, Sector 17-A, Yamuna Expressway, Greater Noida, Gautam Buddh Nagar, U.P., India

Operating Systems

Course code: R1UC403B

B.Tech. (*4th Sem.2023-2024*)

Assignment 1

Submitted by:

Name: **Rohit Pal**

Admn No: 22SCSE1180169

Submitted to:

Mrs. R. Sathiya Priya

Date: 31-03-2024

Q1. Enumerate the basic functions of operating system and explain each in brief.

A1. Operating systems serve as a crucial intermediary between computer hardware and software applications, providing a range of fundamental functions to ensure efficient and secure operation. Here are the basic functions of an operating system, along with brief explanations of each:

1. Process Management:

- The OS manages processes, which are running instances of programs, allocating resources such as CPU time, memory, and I/O devices efficiently.

- It oversees the creation, termination, suspension, and resumption of processes, ensuring fair utilization of system resources and preventing conflicts between processes.

2. Memory Management:

- This function involves managing the computer's memory hierarchy, which includes RAM, virtual memory, and secondary storage (e.g., hard drives).

- The OS allocates memory to processes as needed, ensures memory protection (preventing one process from accessing another's memory), and handles memory swapping to optimize system performance.

3. File System Management:

- Operating systems provide a hierarchical structure for organizing and storing files on storage devices such as hard drives and SSDs.

- They manage file creation, deletion, copying, renaming, and moving, as well as access control through permissions and security mechanisms.

4. Device Management:

- This function involves managing input/output (I/O) devices such as keyboards, mice, printers, network interfaces, and storage devices.

- The OS coordinates communication between applications and hardware devices, handles device drivers, and ensures efficient utilization of devices through techniques like buffering and caching.

5. Security:

- Operating systems implement security mechanisms to protect the system, user data, and applications from unauthorized access, malicious software, and other threats.

- They enforce access control policies, authenticate users, encrypt sensitive data, and provide tools for configuring and monitoring security settings.

6. User Interface:

- The OS provides a user interface (UI) through which users interact with the computer system.

- This can include graphical user interfaces (GUIs) with windows, icons, menus, and pointers (WIMP), or command-line interfaces (CLIs) where users type commands to perform tasks.

7. Networking:

- Modern operating systems include networking capabilities to enable communication between computers and devices over local area networks (LANs) or the internet.

- They support protocols for data transmission, address assignment, routing, and network security, allowing users to access remote resources and services.

8. Error Handling:

- Operating systems handle errors and exceptions that may occur during system operation, such as hardware faults, software crashes, or invalid user inputs.

- They provide mechanisms for detecting, reporting, and recovering from errors to maintain system stability and reliability.

These functions collectively enable an operating system to manage hardware resources efficiently, provide a stable and secure execution environment for applications, and facilitate user interaction with the computer system.

Q2. Enumerate various OS components and give their functions in brief.

A2. Operating systems are composed of several components, each with its specific function to ensure the proper functioning of the system. Here are the main components of an operating system along with brief explanations of their functions:

1. Kernel:

- The kernel is the core component of the operating system, responsible for managing system resources and providing essential services to other parts of the OS and user applications.

- It controls CPU scheduling, memory management, device I/O, and other critical functions required for the operation of the system.

2. Device Drivers:

- Device drivers are software modules that enable the operating system to communicate with hardware devices such as printers, keyboards, mice, and network interfaces.

- They provide an interface between the hardware devices and the kernel, allowing the OS to control and utilize these devices effectively.

3. Shell:

- The shell is the user interface through which users interact with the operating system, issuing commands and executing programs.
- It can be either a command-line interface (CLI), where users type commands, or a graphical user interface (GUI), which provides visual elements such as windows, icons, and menus for user interaction.

4. File System:

- The file system is responsible for organizing and managing files stored on storage devices such as hard drives, SSDs, and USB drives.
- It provides a hierarchical structure for organizing files and directories, handles file operations such as creation, deletion, and modification, and implements access control mechanisms to protect file integrity and privacy.

5. Process Management Component:

- This component manages processes, which are running instances of programs, on the system.
- It oversees process creation, scheduling, termination, and synchronization, ensuring efficient utilization of CPU resources and preventing conflicts between processes.

6. Memory Management Component:

- Memory management component handles the allocation and deallocation of memory resources in the system.
- It manages physical memory (RAM) and virtual memory, ensuring that processes have adequate memory space to execute and implementing techniques such as paging and segmentation to optimize memory usage.

7. I/O Management Component:

- The I/O management component is responsible for managing input and output operations between the computer system and external devices.
- It controls device access, handles data transfer between devices and memory, and implements buffering and caching mechanisms to improve I/O performance.

8. Networking Component:

- This component enables communication between multiple computer systems and devices over networks.
- It supports networking protocols, manages network connections, and provides services such as IP address assignment, routing, and firewall protection.

9. Security Component:

- The security component implements security mechanisms to protect the system, user data, and applications from unauthorized access and malicious attacks.

- It enforces access control policies, encrypts sensitive data, detects and prevents security threats, and provides tools for configuring security settings.

These components work together to provide a stable, efficient, and secure environment for running applications and managing system resources in an operating system.

Q3. Differentiate among multiprogramming, multi-tasking and multiprocessing operating system.

A3. While multiprogramming, multi-tasking, and multiprocessing are all techniques used in operating systems to enhance system performance and efficiency, they differ in terms of their core principles and how they manage the execution of tasks. Here's a breakdown of each:

1. Multiprogramming:

- Definition: Multiprogramming refers to the ability of an operating system to execute multiple programs simultaneously by loading several programs into memory and switching between them.

- Core Principle: The primary goal of multiprogramming is to maximize CPU utilization by keeping the CPU busy with executing one program while another program is waiting for I/O operations or other tasks.

- Execution Model: In multiprogramming, programs are loaded into memory concurrently, and the CPU switches rapidly between them, giving the illusion of concurrent execution.

- Example: Batch processing systems, where multiple jobs are loaded into memory and executed one after another, with the CPU switching between them as needed.

2. Multi-tasking:

- Definition: Multi-tasking refers to the ability of an operating system to execute multiple tasks or processes concurrently, allowing users to run multiple applications simultaneously.

- Core Principle: The primary goal of multi-tasking is to provide a responsive and interactive user experience by allowing users to switch between applications seamlessly.

- Execution Model: In multi-tasking, the CPU switches rapidly between multiple running processes, giving each process a time slice or quantum during which it can execute.

- Example: Modern desktop and mobile operating systems where users can run multiple applications concurrently, such as browsing the web, editing documents, and listening to music.

3. Multiprocessing:

- Definition: Multiprocessing refers to the use of multiple processors or CPU cores within a single computer system to execute multiple tasks or processes concurrently.

- Core Principle: The primary goal of multiprocessing is to improve system performance and scalability by parallelizing the execution of tasks across multiple processors.

- Execution Model: In multiprocessing, tasks or processes are distributed across multiple processors, allowing them to execute simultaneously and independently.

- Example: High-performance computing systems, servers, and modern desktop computers with multiple CPU cores where tasks can be executed in parallel, leading to faster processing speeds.

In summary, while multiprogramming focuses on maximizing CPU utilization by switching between multiple programs in memory, multi-tasking enables users to run multiple applications concurrently for a responsive user experience, and multiprocessing leverages multiple CPU cores to execute tasks in parallel for improved performance and scalability.

Q4. What is interactive operating system?

A4. An interactive operating system is a type of operating system designed to facilitate direct interaction between users and the computer system through a user interface (UI). The primary characteristic of an interactive operating system is its responsiveness to user input, allowing users to interact with the system in real-time and receive immediate feedback.

Key features of interactive operating systems include:

1. User Interface: Interactive operating systems provide user interfaces that allow users to input commands, manipulate files, launch applications, and perform various tasks. These interfaces can be graphical, such as windows, icons, menus, and pointers (WIMP), or command-line interfaces (CLIs) where users type commands.

2. Real-time Response: Interactive operating systems prioritize responsiveness to user input, providing immediate feedback to user actions. This ensures that users can interact with the system in real-time without noticeable delays or lags.

3. Multi-tasking: Interactive operating systems support multi-tasking, allowing users to run multiple applications simultaneously and switch between them seamlessly. This enables users to perform multiple tasks concurrently without interrupting their workflow.

4. Resource Management: Interactive operating systems manage system resources efficiently to ensure smooth operation and responsiveness. This includes CPU scheduling, memory management, and I/O management to optimize performance and maintain a responsive user experience.

5. Error Handling: Interactive operating systems provide mechanisms for error handling and recovery to address unexpected errors or issues that may arise during system operation. This helps maintain system stability and prevents disruptions to user interaction.

Examples of interactive operating systems include modern desktop operating systems such as Windows, macOS, and various Linux distributions, as well as mobile operating systems like iOS and Android. These operating systems provide intuitive user interfaces, support multi-tasking, and offer real-time responsiveness to user input, making them suitable for interactive computing environments.

Q5. Define kernel and explain different types of kernels.

A5. The kernel is the core component of an operating system (OS) that provides essential services and functions to enable communication between hardware and software components. It acts as an intermediary layer, facilitating interactions between user applications and the underlying hardware resources of a computer system. The kernel manages various aspects of system operation, including process management, memory management, device drivers, file system access, and security.

Different types of kernels can be categorized based on their architectural design and how they handle system calls and hardware interactions. The main types of kernels include:

1. Monolithic Kernel:

- In a monolithic kernel architecture, all OS services and functionalities are implemented as a single, large binary executable running in kernel mode.

- System services such as process management, memory management, file system access, and device drivers are tightly integrated into the kernel.

- Monolithic kernels offer efficient performance as system calls and operations incur minimal overhead due to direct access to kernel data structures.

- Examples of operating systems with monolithic kernels include Linux, Unix, and early versions of Windows (e.g., Windows 95 and Windows 98).

2. Microkernel:

- A microkernel architecture follows a minimalist approach, where only essential OS services, such as inter-process communication (IPC) and memory management, are implemented in the kernel.

- Other system services, such as device drivers and file system access, are implemented as user-space processes or modules running outside the kernel in user mode.

- Microkernel designs aim to improve system stability, security, and modularity by minimizing the complexity of the kernel and delegating non-essential functionalities to user-space components.

- Examples of operating systems with microkernels include QNX and MINIX.

3. Hybrid Kernel:

- Hybrid kernels combine elements of both monolithic and microkernel architectures, incorporating some kernel-mode services directly into the kernel while implementing others as separate user-space processes or modules.

- Hybrid kernels aim to strike a balance between the performance benefits of monolithic kernels and the flexibility and modularity of microkernels.

- Operating systems with hybrid kernels may offer a wide range of system services running in both kernel mode and user mode, depending on the design goals and requirements.

- Examples of operating systems with hybrid kernels include Microsoft Windows NT family (e.g., Windows 10) and macOS.

Each type of kernel architecture has its advantages and trade-offs in terms of performance, stability, security, and flexibility, and the choice of kernel design often depends on the specific requirements and goals of the operating system.

Q6. Explain different services provided by operating system.

A6. Operating systems provide a variety of services to manage hardware resources efficiently, facilitate communication between software applications and hardware components, and ensure a stable and secure computing environment. Here are the main services provided by operating systems:

1. Process Management:

- The OS manages processes, which are running instances of programs, by allocating CPU time, memory, and other resources.

- It oversees the creation, termination, suspension, and resumption of processes, ensuring fair utilization of system resources and preventing conflicts between processes.

2. Memory Management:

- This service involves managing the computer's memory hierarchy, including RAM, virtual memory, and secondary storage (e.g., hard drives).

- The OS allocates memory to processes as needed, ensures memory protection (preventing one process from accessing another's memory), and handles memory swapping to optimize system performance.

3. File System Management:

- Operating systems provide a hierarchical structure for organizing and storing files on storage devices such as hard drives and SSDs.

- They manage file creation, deletion, copying, renaming, and moving, as well as access control through permissions and security mechanisms.

4. Device Management:

- This service involves managing input/output (I/O) devices such as keyboards, mice, printers, network interfaces, and storage devices.

- The OS coordinates communication between applications and hardware devices, handles device drivers, and ensures efficient utilization of devices through techniques like buffering and caching.

5. Security Services:

- Operating systems implement security mechanisms to protect the system, user data, and applications from unauthorized access, malicious software, and other threats.

- They enforce access control policies, authenticate users, encrypt sensitive data, and provide tools for configuring and monitoring security settings.

6. User Interface Services:

- The OS provides user interfaces (UIs) through which users interact with the computer system.

- This can include graphical user interfaces (GUIs) with windows, icons, menus, and pointers (WIMP), or command-line interfaces (CLIs) where users type commands to perform tasks.

7. Networking Services:

- Modern operating systems include networking capabilities to enable communication between computers and devices over local area networks (LANs) or the internet.

- They support protocols for data transmission, address assignment, routing, and network security, allowing users to access remote resources and services.

8. Error Handling Services:

- Operating systems handle errors and exceptions that may occur during system operation, such as hardware faults, software crashes, or invalid user inputs.

- They provide mechanisms for detecting, reporting, and recovering from errors to maintain system stability and reliability.

These services collectively enable an operating system to manage hardware resources efficiently, provide a stable and secure execution environment for applications, and facilitate user interaction with the computer system.

Q7. What are the interrupts? How are they handled by the operating system.

A7. Interrupts are signals sent to the CPU by hardware devices or software to request immediate attention or indicate the occurrence of an event that requires handling by the operating system. Interrupts can be generated by various sources, including hardware devices (such as input/output devices or timers) and software conditions (such as system calls or exceptions).

Here's how interrupts are handled by the operating system:

1. Interrupt Detection:

- The CPU constantly monitors for interrupt signals from hardware devices or software conditions.

- When an interrupt occurs, the CPU temporarily suspends the execution of the current instruction and acknowledges the interrupt request.

2. Interrupt Handling:

- Upon detecting an interrupt, the CPU transfers control to a specific location in memory known as the interrupt handler or interrupt service routine (ISR).

- The interrupt handler is a piece of code provided by the operating system to handle the interrupt. It performs tasks such as saving the current state of the CPU, determining the source of the interrupt, and initiating appropriate actions to handle the interrupt.

3. Interrupt Service Routine (ISR):

- The interrupt service routine executes the necessary actions to respond to the interrupt. This may involve servicing the hardware device that generated the interrupt, handling system calls or exceptions, or performing other tasks as required.

- Once the interrupt has been serviced, the CPU restores the saved state and resumes execution of the interrupted program or task.

4. Interrupt Priority and Nesting:

- Interrupts can have different priorities, and the operating system typically assigns priorities to ensure that higher-priority interrupts are handled promptly.

- In some systems, interrupts can be nested, meaning that an interrupt handler may itself be interrupted by a higher-priority interrupt. In such cases, the CPU must handle interrupt nesting properly to prevent data corruption or loss of information.

5. Interrupt Masking:

- The operating system may temporarily disable interrupts (interrupt masking) to prevent interrupt handlers from being interrupted by lower-priority interrupts during critical sections of code execution.

- Interrupt masking helps maintain the integrity of data and prevent race conditions in multi-threaded or multi-tasking environments.

Overall, interrupts play a crucial role in allowing the operating system to respond to events and manage hardware devices efficiently. By handling interrupts promptly and appropriately, the operating system ensures the smooth operation and proper functioning of the computer system.

Q8. Describe the essential properties of the following operating system: (a) Real time and (b)Distributed operating system.

A8.

(a) Real-time Operating System (RTOS):

1. Predictability: RTOSs prioritize predictability in the timing of responses to events. They guarantee that critical tasks will meet their deadlines within specified time constraints. This predictability is crucial for systems where timely responses are critical, such as industrial control systems, automotive systems, and medical devices.

2. Task Scheduling: RTOSs use deterministic scheduling algorithms to ensure that tasks with higher priority are executed before tasks with lower priority. This scheduling approach guarantees that time-critical tasks are completed within their deadlines. Common scheduling algorithms in RTOSs include Rate Monotonic Scheduling (RMS) and Earliest Deadline First (EDF).

3. Minimal Latency: RTOSs aim to minimize the latency between the occurrence of an event and the response by the system. Low-latency is essential for real-time applications to ensure that responses to external stimuli occur within a specified time frame.

4. Interrupt Handling: RTOSs prioritize interrupt handling to ensure that time-sensitive events, such as sensor inputs or external signals, are processed promptly. Interrupt latency—the time taken to respond to an interrupt—is minimized to meet stringent real-time requirements.

5. Resource Management: RTOSs manage system resources efficiently to guarantee that critical tasks have access to the necessary CPU time, memory, and I/O resources. Resource allocation and scheduling decisions are made with the goal of meeting timing constraints and minimizing contention.

6. Fault Tolerance: Many RTOSs incorporate fault-tolerant mechanisms to ensure system reliability. This may include redundancy in critical components, error detection and recovery mechanisms, and the ability to maintain system operation even in the presence of faults or failures.

(b) Distributed Operating System:

1. Transparency: Distributed operating systems aim to provide transparency to users and applications, hiding the complexities of distributed computing. Transparency includes location transparency (hiding the physical location of resources), access transparency (providing uniform access methods), and failure transparency (masking failures).

2. Concurrency: Distributed operating systems support concurrent execution of processes across multiple nodes in the distributed system. They manage concurrency through mechanisms such as distributed mutual exclusion, distributed locking, and distributed concurrency control protocols.

3. Communication: Communication is fundamental in distributed operating systems. They provide mechanisms for inter-process communication (IPC) and network communication to enable processes running on different nodes to exchange data and coordinate their actions. Common communication paradigms include message passing, remote procedure calls (RPC), and distributed shared memory.

4. Scalability: Distributed operating systems are designed to scale horizontally, allowing the addition of new nodes to the system to accommodate increased workload or resource demands. Scalability is achieved through load balancing, distributed resource management, and dynamic reconfiguration.

5. Fault Tolerance: Distributed operating systems incorporate fault-tolerant mechanisms to ensure system reliability despite the presence of failures or faults in the distributed environment. This may include replication of critical data and services, distributed error detection and recovery mechanisms, and fault-tolerant communication protocols.

6. Security: Security is paramount in distributed operating systems to protect sensitive data and ensure the integrity, confidentiality, and availability of resources in the distributed environment. They implement authentication, access control, encryption, and other security mechanisms to mitigate security threats and vulnerabilities.

Both real-time and distributed operating systems have unique properties tailored to their specific requirements and environments, allowing them to meet the diverse needs of real-time applications and distributed computing scenarios.

Q9. Explain the architecture of different operating system structure?

A9. Operating systems can be structured in various architectures, each with its own design principles and organization of components. Here are some common operating system architectures:

1. Monolithic Kernel:

- In a monolithic kernel architecture, all operating system services run in kernel mode, within the same address space.

- The kernel provides all system services, including process management, memory management, file system access, and device drivers.

- Monolithic kernels offer efficient performance as system calls and operations incur minimal overhead due to direct access to kernel data structures.

- Examples include early versions of Unix, Linux, and Windows (e.g., Windows 95 and Windows 98).

2. Layered Architecture:

- In a layered architecture, the operating system is divided into layers, with each layer providing a set of related functions and services.

- Each layer communicates only with adjacent layers, following a strict hierarchy.

- Common layers include hardware abstraction layer (HAL), kernel layer, device management layer, file system layer, and user interface layer.

- Layered architectures promote modularity and ease of maintenance but may suffer from performance overhead due to inter-layer communication.

- Examples include the THE, Multics, and the OSI model (not strictly an operating system but follows a layered architecture).

3. Microkernel:

- A microkernel architecture follows a minimalist approach, where only essential OS services, such as inter-process communication (IPC) and memory management, are implemented in the kernel.

- Other system services, such as device drivers and file system access, are implemented as user-space processes or modules running outside the kernel in user mode.

- Microkernel designs aim to improve system stability, security, and modularity by minimizing the complexity of the kernel and delegating non-essential functionalities to user-space components.

- Examples include MINIX and QNX.

4. Hybrid Kernel:

- Hybrid kernels combine elements of both monolithic and microkernel architectures, incorporating some kernel-mode services directly into the kernel while implementing others as separate user-space processes or modules.

- Hybrid kernels aim to strike a balance between the performance benefits of monolithic kernels and the flexibility and modularity of microkernels.

- Operating systems with hybrid kernels may offer a wide range of system services running in both kernel mode and user mode.

- Examples include Microsoft Windows NT family (e.g., Windows 10) and macOS.

5. Virtual Machine Architecture:

- In a virtual machine architecture, the operating system creates a layer of abstraction between the hardware and applications by running multiple virtual machines (VMs), each with its own operating system instance.

- The hypervisor or virtual machine monitor (VMM) manages the virtualization of hardware resources and enables the execution of multiple operating systems on a single physical machine.

- Virtual machine architectures provide isolation between different operating system instances and offer flexibility in running diverse software environments.

- Examples include VMware ESXi, Microsoft Hyper-V, and KVM (Kernel-based Virtual Machine).

Each architecture has its advantages and trade-offs in terms of performance, security, modularity, and complexity, and the choice of architecture often depends on the specific requirements and goals of the operating system.

Q10. What is the difference between hard and soft real time system?

A10. Hard real-time systems and soft real-time systems are both types of real-time systems but differ in their tolerance for missing deadlines and the consequences of such misses. Here are the key differences between hard real-time systems and soft real-time systems:

1. Deadline Strictness:

- Hard Real-Time Systems: In hard real-time systems, meeting deadlines is critical. Failure to complete a task within its specified deadline can lead to catastrophic consequences, such as system failure, safety hazards, or financial losses. Thus, hard real-time systems must guarantee that tasks are completed within their deadlines with high reliability.

- Soft Real-Time Systems: Soft real-time systems have less stringent deadline requirements. While meeting deadlines is still desirable, occasional misses may be tolerable without catastrophic consequences. The primary goal in soft real-time systems is to ensure that the majority of tasks are completed within their deadlines, but occasional deadline misses are acceptable as long as they do not significantly impact system performance or functionality.

2. Consequences of Deadline Misses:

- Hard Real-Time Systems: Deadline misses in hard real-time systems can have severe consequences, including system failure, loss of critical data, safety hazards, or even loss of human life. These systems prioritize determinism and predictability to ensure that tasks are executed within their deadlines reliably.

- Soft Real-Time Systems: Deadline misses in soft real-time systems may lead to degraded performance or reduced quality of service but are generally tolerable as long as they do not jeopardize system integrity or safety. Soft real-time systems often focus on maximizing throughput or optimizing resource utilization while still striving to meet deadlines for most tasks.

3. Design Trade-offs:

- Hard Real-Time Systems: Designing hard real-time systems requires careful consideration of worst-case execution times, deterministic scheduling algorithms, and resource allocation strategies to ensure that deadlines are met consistently. These systems often prioritize simplicity, reliability, and real-time predictability over flexibility and optimization.

- Soft Real-Time Systems: Soft real-time systems may prioritize factors such as responsiveness, throughput, and efficiency over strict deadline adherence. Designing soft real-time systems involves balancing deadline requirements with other system objectives, such as maximizing system utilization or optimizing user experience.

4. Examples:

- Hard Real-Time Systems: Examples of hard real-time systems include automotive safety systems (e.g., anti-lock braking systems), avionics systems, medical devices (e.g., pacemakers), and industrial control systems (e.g., robotics in manufacturing).

- Soft Real-Time Systems: Examples of soft real-time systems include multimedia streaming applications, online gaming, financial trading systems, and interactive user interfaces.

In summary, the primary difference between hard real-time systems and soft real-time systems lies in their tolerance for deadline misses and the severity of consequences associated with such misses. Hard real-time systems require strict adherence to deadlines to prevent catastrophic failures, while soft real-time systems can tolerate occasional deadline misses without significant adverse effects.