Operating System

Bachelors in Computer Engineering

Chapter Seven

New Trends in Operating System

OUTLINES:

- Concept, character and role of
- Real-time Operating System,
- Distributed Operating System
- Cloud Operating System
- Mobile Operating System
- Security issues and method of deployment
- Memory wall and bottleneck for Operating system

Real-time Operating System

- A Real-Time Operating System (RTOS) is an operating system specifically designed to meet the requirements of real-time systems. Real-time systems are those in which the correctness of the system depends not only on the logical result of computation but also on the time at which the results are produced.
- A Real-Time Operating System plays a critical role in ensuring that tasks in a real-time system are executed within specified time constraints, providing determinism, low latency, and reliability. It is a fundamental component in applications where timing and responsiveness are crucial, such as in industrial automation, medical devices, and automotive control systems.

- **Hard Real-Time:**
- In hard real-time systems, meeting deadlines is crucial. If a task or process misses its deadline, it can lead to system failure. Examples include aerospace systems, medical equipment, and automotive safety systems.
- An automotive airbag system is a classic example of a hard real-time system. When a collision is detected, the airbag must deploy within a very strict and deterministic timeframe to ensure passenger safety.

- **Soft Real-Time:**
- Soft real-time systems have less stringent timing constraints. Missing a deadline may not lead to catastrophic consequences but can degrade system performance. Examples include multimedia applications and video streaming.
- A video streaming service is an example of a soft real-time system. While it's desirable to minimize latency, missing occasional deadlines (such as buffering or a slight delay in video playback) might not have severe consequences.

Characteristics of RTOS:

- **Deterministic Behavior:** RTOS is designed to provide deterministic behavior, where the time taken to perform a specific task is predictable and consistent.
- **Low Latency:** RTOS minimizes the time between an event occurrence and the system's response to that event.
- **Task Scheduling:** RTOS uses priority-based or time-based scheduling algorithms to ensure that critical tasks are executed on time.
- **Reliability:** RTOS must be highly reliable to ensure that critical tasks are executed without failures.

- **Task:** In RTOS, tasks represent individual units of work that need to be performed. These tasks are scheduled and executed based on priority or a predefined schedule.
- **Processes:** Processes are similar to tasks but may have additional characteristics, such as separate memory spaces. RTOS manages processes efficiently to ensure real-time requirements are met.

Role of Real-Time OS:

1. **Task Management:** RTOS manages tasks to ensure they are executed in a timely manner. It handles task scheduling, priority management, and synchronization.
2. **Interrupt Handling:** RTOS efficiently handles interrupts to respond quickly to external events. Interrupt latency (the time between the occurrence of an interrupt and the start of its processing) is minimized.
3. **Resource Management:** RTOS manages system resources such as CPU, memory, and peripherals to ensure that tasks meet their deadlines.
4. **Communication and Synchronization:** RTOS provides mechanisms for inter-task communication and synchronization, allowing tasks to exchange information and coordinate their actions.

Distributed Operating System:

- A distributed system is the collection of independent computers that appears to the user of the system as a single computer. It is a collection of processors that do not share memory or a clock.
- Instead, each processor has its own local memory. The processors communicate with one another through various communication networks, such as high speed buses or telephone lines.
- The processors in a distributed system may vary in size and function. These processors are also known as sites, nodes, computers, machines, and hosts.
- A distributed system has:
  - Multiple connected CPUs working together
  - A collection of independent computers that appears to its users as a single coherent system
- Example:
  - Let us suppose a large bank with hundreds of branch offices all over the world. Each office has a master computer to store local accounts and handle local transactions. In addition, each computer has the ability to talk to all other branch computers and with a central computer at headquarters.
  - If transactions can be done without regard to where a customer or account is, and the users do not notice any difference between this system and the old centralized mainframe that it replaced, it too would be considered a distributed system.

Characteristics of Distributed Operating Systems:

Concurrency:

Multiple processes can execute concurrently on different machines. Coordination mechanisms are required to manage concurrency.

Transparency:

Users and applications are shielded from the complexities of the distributed nature of the system. Transparency includes access, location, migration, relocation, replication, and failure.

Scalability:

The system can easily expand by adding more machines, and the performance should ideally improve linearly with the addition of resources.

Reliability:

Distributed systems often have multiple points of failure, so reliability measures, such as redundancy and fault tolerance, are crucial.

Heterogeneity:

Different machines with varying hardware and software configurations can be part of the distributed system.

Security:

Security mechanisms must be in place to ensure the confidentiality and integrity of data, especially when it's transmitted across different machines.

Roles of Distributed Operating Systems:

Resource Management:

Efficient allocation and management of resources such as processors, memory, and storage across multiple machines.

Communication:

Establishing communication channels between processes running on different machines. This involves designing communication protocols and ensuring reliable data transfer.

Synchronization:

Managing the order of execution of processes to avoid conflicts and ensure consistency in a distributed environment.

Fault Tolerance:

Implementing mechanisms to detect, isolate, and recover from failures to ensure the continued operation of the system.

Naming and Directory Services:

Managing distributed naming and directory services to provide a consistent and transparent view of the resources in the system.

Security:

Enforcing security measures to protect data and communication within the distributed system.

Distributed File Systems:

Providing access to files and storage resources in a transparent and efficient manner across the distributed environment.

Middleware:

Offering middleware services that abstract low-level details of communication, allowing easier development of distributed applications.

•

Advantages:

- Resource sharing:
  - In resource sharing system a number of different sites are connected to one other. So a user at one site may be able to use the resources available at another.
- Computation speedup:
  - Distributed system can perform load sharing. In such system a particular computation can be partitioned into sub computations that can run concurrently and then it allows us to distribute the sub computations among the various sites to run concurrently. Thus this system increases computation speed.
- Reliability:
  - In such system, if one site in the distributed system fails then the remaining sites can continue operating their task.
- Communication:
  - In this system several sites are connected through communication networks through which users can exchange information. At a low level, messages (such as file transfer, remote login, mail, remote procedure calls, etc) are passed between systems.

Disadvantages:

- Complexity:
  - It is complex system software from the user, designers view. It is harder to understand what will happen at any given case and even harder to design software to handle even understood complexities. It also requires more complex synchronization.
- Difficulties of allocating resources:
  - In this system, local machine may have inadequate resources for a task. It may arise resource conflicts while using remote resources.
- Security:
  - There is no centralized control. So there is lack of security problem.
- Heterogeneity Problems:
  - Such system consists of different kinds of heterogeneous systems (hardware and software). So there might be problem with data formats, executable formats, software versioning, and different operating system.

Cloud Operating System

- Cloud computing refers to the delivery of computing services over the internet, allowing users to access and use resources such as servers, storage, databases, networking, software, analytics, and more, without the need for owning or managing the physical infrastructure.
- Cloud computing is characterized by on-demand availability, scalability, and resource pooling.

Characteristics of Cloud Computing:

On-Demand Self-Service: Users can provision and manage computing resources as needed, without human intervention from the service provider.

Broad Network Access: Services are available over the network and can be accessed through standard mechanisms by diverse client devices (e.g., laptops, smartphones).

Resource Pooling: Computing resources are pooled to serve multiple users, with different physical and virtual resources dynamically assigned and reassigned according to demand.

Rapid Elasticity: Resources can be rapidly scaled up or down to accommodate changes in demand, providing flexibility and cost efficiency.

Measured Service: Cloud computing resources are metered, and users pay for only what they consume. This pay-as-you-go model enhances cost-effectiveness.

Role of Cloud Computing in Operating Systems:

- While the term "Cloud Operating System" might not be widely used, cloud computing heavily influences the way operating systems function and are utilized. Key aspects include:
- **Virtualization:**
  - Cloud computing often relies on virtualization technologies to create virtual instances of operating systems and applications.
  - This enables efficient resource utilization and isolation.
- **Resource Management:**
  - Cloud providers employ operating systems that efficiently manage resources across a vast number of virtual machines or containers.
  - Resource allocation, load balancing, and isolation are crucial aspects.

- **Service Orchestration:**
  - Operating systems play a role in orchestrating various cloud services, ensuring they work together seamlessly.
  - This includes tasks like provisioning, configuration, and scaling of resources.
- **Security and Compliance:**
  - Cloud operating systems must address security concerns related to data integrity, confidentiality, and availability.
  - Compliance with industry standards and regulations is essential.
- **Networking:**
  - Operating systems within the cloud context need to handle networking aspects efficiently, including communication between virtual machines, load balancing, and ensuring secure data transfer.

Mobile Operating System

- A mobile operating system (OS) is a specialized operating system designed to run on mobile devices such as smartphones, tablets, and other handheld devices.
- It facilitates the interaction between the hardware of a mobile device and the applications that run on it. It provides essential services, manages hardware resources, and enables users to run applications on their mobile devices.
- Mobile operating systems are optimized for power efficiency, touch input, and small form factors.
- Popular examples of mobile operating systems include Android, iOS. Each of these platforms has its unique characteristics and plays a crucial role in shaping the mobile user experience.

Characteristics of Mobile Operating Systems:

Touchscreen Support:

Mobile operating systems are designed to support touch input as a primary method of interaction. Gestures like tapping, swiping, and pinching are integral to the user experience.

Resource Efficiency:

Mobile devices have limited resources compared to desktop computers, so mobile operating systems are optimized for resource efficiency, including battery life, CPU usage, and memory management.

App Ecosystem:

Mobile operating systems typically have centralized app stores where users can download and install applications. These ecosystems are curated, providing a level of security and control.

Multitasking:

Mobile operating systems allow for multitasking, allowing users to switch between different applications seamlessly. However, due to resource constraints, mobile OSes often manage background processes more aggressively.

Security:

Security is a critical aspect of mobile operating systems. They incorporate features like app sandboxing, secure boot, encryption, and permission systems to protect user data and the device itself.

Connectivity:

Mobile OSes provide extensive support for wireless connectivity, including Wi-Fi, cellular data, Bluetooth. This connectivity is crucial for features like internet browsing, messaging, and file sharing.

Role of Mobile Operating Systems:

Hardware Abstraction:

Mobile operating systems abstract the underlying hardware complexities, allowing application developers to write software that can run on a variety of devices with different hardware specifications.

User Interface:

Mobile OSes provide the graphical user interface (GUI) that users interact with on their devices. This includes the home screen, app launcher, notification center, and settings.

App Management:

Mobile operating systems manage the installation, update, and removal of applications. They also provide mechanisms for organizing and accessing installed apps.

Device Configuration:

Mobile OSes allow users to configure various settings on their devices, such as network settings, display preferences, security options, and more.

Security and Privacy:

Mobile operating systems implement security measures to protect against malware, unauthorized access, and data breaches. They also give users control over app permissions and privacy settings.

Updates and Upgrades:

Mobile OS providers regularly release updates and upgrades to add new features, improve performance, and address security vulnerabilities.

Security issues and method of deployment

Security best practices for OS deployment

- Implement access controls to protect bootable media
- Use a secure location when you create media for OS images
- Protect certificate files
- Block or revoke any compromised certificates
- Secure the communication channel between the site server and the SMS Provider
- Enable distribution points for PXE client communication only on secure network segments
- Configure PXE-enabled distribution points to respond to PXE requests only on specified network interfaces
- Require a password to PXE boot
- Restrict content in OS images used for PXE boot or multicast
- Restrict content installed by task sequence variables
- Secure the network channel when migrating user state
- Use the latest version of USMT
- Manually delete folders on state migration points when you decommission them

- Don't configure the deletion policy to immediately delete user state
- Manually delete computer associations
- Manually back up the user state migration data on the state migration point
- Implement access controls to protect the prestaged media
- Implement access controls to protect the reference computer imaging process
- Always install the most recent security updates on the reference computer
- Implement access controls when deploying an OS to an unknown computer
- Enable encryption for multicast packages
- Monitor for unauthorized multicast-enabled distribution points
- When you export task sequences to a network location, secure the location and secure the network channel
- If you use the task sequence run as account, take additional security precautions
- Restrict and monitor the administrative users who are granted the OS deployment manager security role
- Use Enhanced HTTP to reduce the need for a network access account

Security issues for OS deployment

- Information disclosure and denial of service
- Impersonation and elevation of privileges
- Client authentication to the state migration point is achieved by using a Configuration Manager token that is issued by the management point.
- If you use collection variables, local administrators can read potentially sensitive information

https://learn.microsoft.com/en-us/mem/configmgr/osd/plan-design/security-and-privacy-for-operating-system-deployment#:~:text=Security%20issues%20for%20OS%20deployment&text=This%20process%20might%20include%20formatting,domain%20and%20volume%20licensing%20keys

Memory wall and bottleneck for Operating system

- **Memory wall** refers to the growing disparity between the speed of the CPU (central processing unit) and the speed of memory (RAM - Random Access Memory).
- As CPUs have become faster over time, the speed at which data can be accessed from memory has not increased at the same rate.
- This results in a performance gap, and the CPU often spends a significant amount of time waiting for data to be fetched from memory.
- This creates a bottleneck where the CPU spends a significant amount of time waiting for data to be fetched from or written to the memory.

- A bottleneck in an operating system refers to a point in the system where the performance is limited or constrained, preventing the system from achieving higher overall performance.
- Memory-related bottlenecks are common due to the memory wall issue, but bottlenecks can occur in other areas as well.

Memory Bottleneck Causes:

High Memory Usage:

Operating systems and applications that consume a large amount of memory can lead to memory bottlenecks, especially in systems with limited RAM.

Inefficient Memory Management:

Poor memory management strategies, such as inefficient memory allocation or excessive swapping between RAM and disk, can contribute to bottlenecks.

Insufficient RAM:

Inadequate physical RAM can lead to excessive use of virtual memory (paging to disk), resulting in a slowdown of system performance.

Contended Access:

Multiple processes or threads contending for access to the same memory resources can create bottlenecks, especially in multithreaded applications.

Addressing Memory Bottlenecks:

Optimizing Memory Usage:

Efficient coding practices and optimizing algorithms can reduce the overall memory footprint of applications.

Improving Memory Management:

Operating systems can implement advanced memory management techniques, such as better caching strategies and improved virtual memory algorithms.

Adding More RAM:

Increasing the physical RAM in a system can help alleviate memory bottlenecks by providing more space for active processes.

Parallelism and Multithreading:

Leveraging parallelism and optimizing code for multithreading can help mitigate bottlenecks by allowing multiple tasks to be executed concurrently.

Use of Faster Memory Technologies:

The adoption of faster memory technologies, such as high-speed RAM or non-volatile memory, can help reduce the impact of the memory wall.

Thank You