ADDIS ABABA UNIVERSITY

COLLEGE OF NATURAL AND COMPUTATIONAL SCIENCES

DEPARTMENT OF COMPUTER SCIENCE

REALTIME AND EMBEDDED SYSTEMS INIDIVIDUAL ASSIGNMENT

BY

NAME:	MIHRETU ENDESHAW
ID:	UGR/4579/12
SECTION:	02

1. What are the characteristics of Real Time and Embedded systems?

- **Time and Resource Constraints:** The system must handle tasks and events with strict timing requirements and resources in terms of processing power, memory, and energy. This includes handling real-time inputs, executing tasks, and generating outputs in a timely manner and the real-time aspect adds the additional challenge of allocating and managing these resources efficiently to meet constraints.
- **Real-time Interactions:** The system may involve real-time interactions between different components or subsystems. These interactions can be data exchange, control signals, or synchronization to achieve coordinated behavior. Proper communication and coordination are essential to maintain timing correctness.
- **Deterministic:** Real-time and embedded systems require deterministic behavior, meaning they must respond to events within specific timing constraints.
- **Predictable Behavior:** The combined system should exhibit consistent and predictable behavior under various conditions and workloads to ensure the requirements real-time.
- **Reliability and Safety:** The system must be designed to handle faults, errors, and exceptions in a manner that guarantees continued operation and minimizes risks.
- **Robustness:** The system needs to be capable of handling unexpected events, variations in workloads, and transient faults and should recover from failures and resume normal operation to maintain operation.

2. Describe Hardware Elements used in RE systems.

Real-time embedded systems consist of various hardware elements that are specifically designed to meet the requirements of real-time and embedded applications.

- ➢ **Micro-controllers:** are the heart of many embedded systems. They are IC that combine a microprocessor core, memory, and input/output peripherals on a single chip. Micro-controllers are designed to provide real-time control.
- ➢ **Field Programmable Gate Arrays (FPGAs):**are programmable logic devices that offer a high level of flexibility and customization. They contain an array of programmable logic blocks and configurable interconnects, allowing designers to implement specific functionality in hardware.
- ➢ **Digital Signal Processors (DSPs**): are specialized microprocessors designed for efficient digital signal processing. They are optimized for performing mathematical operations on digital signals and are commonly used in applications such as audio processing, image processing, telecommunications, and control systems.
- ➢ **Sensors and Actuators:** Real-time embedded systems often interact with the physical world through sensors and actuators. Sensors capture environmental data such as temperature, pressure, motion, light, etc., while actuators control physical processes or devices.
- ➢ **Memory Devices:** ES require various types of memory for program storage, data storage, and temporary storage. These include ROM for storing permanent program code, Flash memory for non-volatile storage, RAM for temporary data storage, and EEPROM for storing configuration data.
- ➢ **Communication Interfaces:** Communication interfaces such as UART (Universal Asynchronous Receiver-Transmitter), SPI, I2C (Inter-Integrated Circuit), Ethernet, USB, and wireless protocols like Wi-Fi, Bluetooth, or Zigbee are used for data exchange and connectivity with other devices.
- ➢ **Power Management Components:** Power management components such as voltage regulators, power switches, and power management units are employed to optimize power consumption, extend battery life, and ensure efficient power distribution for Embedded system.

3. Describe structure of Real Time and Embedded (RE) operating systems clearly specifying difference between Interrupt Driven, Nano-kernel, Micro-kernel and Monolithic kernel.

- **<u>Interrupt-Driven Model:</u>** is centered around interrupts, which are hardware signals that pause program execution to handle urgent events. It prioritizes interrupt handling and responsiveness over modularity, utilizing a single, comprehensive kernel for all system services. The focus is on efficient interrupt management rather than strict separation of kernel and user space.
- **<u>Nanokernel Model:</u>** is a modular approach to operating system design that focuses on minimizing kernel size and functionality. It provides essential services while implementing additional features as separate user-level processes or modules. This emphasis on modularity reduces the kernel's complexity and enhances efficiency.
- **<u>Microkernel Model:</u>** modularity is taken to a greater extent by minimizing the kernel's functionality and shifting most operating system services and drivers to user-level processes or servers. The kernel offers essential services like process scheduling and inter-process communication, while other functionalities such as file systems, network protocols, and device drivers are executed in separate user-level processes. These processes communicate with each other through message passing. This approach promotes strong modularity and service separation, resulting in a smaller, more reliable kernel that is easier to maintain. It also enhances fault isolation and extensibility through separate user-level processes for system services, device drivers, and file systems.
- **<u>Monolithic Kernel Model:</u>**is the traditional approach to operating system design, where all essential components such as device drivers, system libraries, and operating system services are bundled together in a single, large kernel. It offers a wide range of functionalities, including process management, memory management, file systems, device drivers, and network protocols. Unlike other models, the monolithic kernel lacks modularity as all services operate within the kernel, which can impact fault isolation and system reliability. Modifying or extending the kernel is also more challenging due to its size and complexity.

4. Differentiate between Periodic, Aperiodic and Sporadic Tasks. What algorithms are available for their scheduling?

- **<u>Periodic Tasks:</u>** are repetitive tasks with specific timing needs that occur at regular intervals. They have a defined period, indicating the time between consecutive occurrences of the task. These tasks include activities such as collecting real-time data, reading sensors, implementing control loops, and generating periodic signals.
- **<u>Aperiodic Tasks:</u>** are one-time tasks without a predictable timing pattern or regular intervals. They are triggered by specific events or external stimuli and lack inherent periodicity. Aperiodic tasks typically come with deadlines, indicating the maximum time allowed for their execution after being requested. Examples include user inputs, interrupts, event-driven processing, and sporadic requests.
- **<u>Sporadic Tasks:</u>** a type of aperiodic tasks, do not have a fixed timing pattern but have minimum time constraints between occurrences. While they lack regular periodicity, they must adhere to a minimum inter-arrival time between consecutive instances. Each instance of a sporadic task has a deadline, indicating the maximum allowed time for its completion. Examples of sporadic tasks include periodic tasks with varying execution times and tasks that exhibit burst behavior.

Scheduling Algorithms for Periodic, Aperiodic, and Sporadic Tasks:

<u>Periodic Task Scheduling:</u>

- ❖ Rate Monotonic Scheduling (RMS) prioritizes tasks based on their periods, giving higher priority to tasks with shorter periods. It ensures that tasks with shorter periods meet their deadlines, as long as the system is schedulable.
- ❖ Earliest Deadline First (EDF) assigns priorities based on absolute deadlines, with the task having the earliest deadline given the highest priority. If the system is schedulable, EDF guarantees meeting all deadlines. Both

scheduling algorithms prioritize tasks based on different criteria to ensure timely task completion in schedulable systems.

Aperiodic Task Scheduling:

- First-Come, First-Served (FCFS): Executes aperiodic tasks in the order they arrive. It can lead to poor performance when high-priority aperiodic tasks arrive after low-priority tasks.
- Earliest Deadline First (EDF): Can also be used to schedule aperiodic tasks by treating their arrival times as deadlines. Aperiodic tasks are scheduled dynamically based on their deadlines, allowing for efficient utilization of resources.

Sporadic Task Scheduling:

- Sporadic Server: Allocates fixed-time slots to each sporadic task, allowing them to execute within the allocated time. The server replenishes the allocated time for each task after it completes.
- Deferral Server: Similar to sporadic server but allows for dynamic allocation of time slots to sporadic tasks based on their deadlines and resource availability.

5. Describe Energy Aware CPU Scheduling.

Energy-aware CPU scheduling is a strategy for task scheduling that aims to minimize energy usage within a system. It involves optimizing resource allocation and task execution decisions to improve energy efficiency. This approach is crucial for prolonging battery life in mobile devices and reducing energy expenses in data centers, utilizing techniques that prioritize power conservation.

- ✓ **Dynamic Voltage and Frequency Scaling (DVFS):** is a method that modifies the CPU's operating frequency and voltage according to the workload. By decreasing the frequency and voltage during low workload periods, it achieves substantial energy savings. Energy-aware CPU scheduling algorithms leverage DVFS to dynamically align the CPU's frequency and voltage with the workload's needs, optimizing energy consumption accordingly.
- ✓ **Task Consolidation:** involves grouping tasks together to reduce the number of active processing cores. By consolidating tasks onto fewer cores, the remaining idle cores can be put into low-power states or even turned off, leading to energy savings. Energy-aware CPU schedulers analyze the workload and determine optimal core utilization to achieve better energy efficiency.
- ✓ **Sleep Mode and Power Management:** Energy-aware scheduling involves utilizing sleep states or low-power modes of the CPU to save energy during periods of inactivity. Idle tasks or tasks with low priority can be temporarily suspended or put into low-power states, allowing the CPU to operate in a more energy-efficient mode.
- ✓ **Energy Models and Prediction:** Energy-aware schedulers often rely on energy models and prediction algorithms to estimate the energy consumption of different scheduling decisions by considering factors such as CPU frequency, voltage, task characteristics, and power states.
- ✓ **Deadline-Aware Scheduling:** Energy-aware scheduling algorithms may also consider task deadlines in addition to energy efficiency. By meeting task deadlines while minimizing energy consumption, the scheduler ensures both timely execution and energy savings.
- ✓ **Power-Aware Task Migration:** In systems with multiple cores or distributed computing setups, energy-aware scheduling can involve transferring tasks between cores or nodes to distribute the workload evenly and enhance energy efficiency. Task migration aims to consolidate tasks onto a subset of active resources while deactivating the remaining ones.What do you understand by Multi-Processor and Distributed (MPD) systems?

6. Describe Architecture of Operating Systems for such systems.

- ✓ **Multi-Processor Systems:** refers to a computer system that houses multiple processors or cores within a single physical machine, sharing resources like memory and I/O devices. It can be classified into Symmetric Multi-Processor (SMP) and Asymmetric Multi-Processor (AMP) systems. SMP systems consist of identical processors with equal access to shared resources, running a single operating system instance across all processors. Load balancing, resource synchronization, and process/thread scheduling mechanisms are crucial in SMP systems. On the other hand, AMP systems feature processors with distinct roles or capabilities, often running separate operating systems or specialized software. AMP is commonly used in embedded systems for handling specific tasks or responsibilities, relying on message passing or shared memory mechanisms for coordination between processors.
- ✓ **Distributed Systems:** refers to a computer system composed of multiple autonomous computing nodes or machines connected through a network. Each node in a distributed system can have its own processor, memory, and local resources. Unlike multi-processor systems, distributed systems do not share a common memory or bus. Instead, communication between nodes occurs through message passing over the network. Distributed systems are designed to work collaboratively and provide fault tolerance, scalability, and distributed processing capabilities.

Operating System Architecture for MPD Systems:

- ✓ **Process and Thread Management:** MPD operating systems need to manage processes and threads across multiple processors or nodes. This includes process/thread creation, scheduling, load balancing, synchronization, and inter-process communication mechanisms.
- ✓ **Memory Management:** MPD systems require efficient memory management to handle distributed or shared memory across processors/nodes. This involves memory allocation, mapping, protection, and coherence protocols to ensure data consistency.
- ✓ **Communication and Coordination:** In distributed systems, communication and coordination mechanisms are crucial. This includes inter-process communication (IPC) mechanisms like message passing, remote procedure calls (RPCs), and synchronization primitives to ensure consistency and coordination between distributed processes or threads.
- ✓ **Fault Tolerance and Reliability:** MPD operating systems often incorporate fault tolerance mechanisms to handle failures in processors or nodes. This includes error detection, error recovery, replication, and distributed consensus protocols to ensure system reliability.
- ✓ **Distributed File Systems:** In distributed systems, distributed file systems provide a transparent and efficient way to access and manage files distributed across multiple nodes. These file systems provide mechanisms for file sharing, replication, and data consistency.
- ✓ **Security and Access Control:** MPD operating systems need to address security concerns and provide access control mechanisms to protect sensitive data and resources in distributed environments. This includes authentication, encryption, and access control policies.

7 How is Resource sharing and Load Balancing achieved in MPD systems?

Resource sharing and load balancing in MPD (Multi-Processor and Distributed) systems are critical aspects to ensure efficient utilization of computing resources and optimal performance.

Resource Sharing

- ✓ **Shared Memory:** In MPD systems with shared memory, multiple processors can access a common physical memory space, enabling resource sharing through read and write operations. Synchronization mechanisms like locks, semaphores, and barriers are employed to coordinate access to shared resources, maintaining data consistency and avoiding conflicts.

- ✓ **Message Passing:** In distributed MPD systems, resource sharing is achieved through the mechanism of message passing. Nodes or processes communicate by exchanging messages over a network, facilitating the sharing of data, requests, and information. Communication protocols and libraries, such as MPI (Message Passing Interface), are commonly employed to facilitate the message passing process in distributed MPD systems.

Load Balancing

Load balancing aims to distribute the workload evenly across processors or nodes in an MPD system to optimize resource utilization and avoid bottlenecks.

- ➢ **Static Load Balancing:** the workload is divided among processors or nodes at the beginning of the execution based on predetermined criteria. The load distribution remains fixed throughout the execution. Static load balancing techniques can be based on task assignment algorithms, such as round-robin, where tasks are assigned in a cyclic manner to each processor, or based on workload characteristics.
- ➢ **Dynamic Load Balancing:** adjusts the workload distribution during run-time based on the current system state. It involves monitoring the system's load, identifying imbalances, and redistributing tasks or workload accordingly. Dynamic load balancing techniques may consider factors such as processor utilization, task execution times, communication costs, and available resources.
- ➢ **Centralized Load Balancing:** a dedicated central entity, often referred to as a load balancer or scheduler, manages the task distribution across processors or nodes. It collects information about the system's load and makes decisions on workload redistribution. It assigns tasks to idle processors or migrates tasks from overloaded processors to underutilized ones. It requires efficient communication between the load balancer and processors or nodes.
- ➢ **Decentralized Load Balancing:** each processor or node is responsible for making load balancing decisions locally based on its own workload and system information. Processors or nodes exchange load information and cooperate to balance the overall workload.

8 What are the Design and Development Challenges in MPD Operating Systems?

- ❖ **Scalability:** MPD systems are designed to scale up by adding more processors or nodes. However, ensuring scalability in terms of performance, resource management, and coordination is challenging as the system grows. Therefore,efficient inter-process communication mechanisms, and load balancing techniques are essential to handle such problems.
- ❖ **Synchronization and Consistency:** Coordinating access to shared data and maintaining data consistency across distributed processors or nodes require careful design and implementation of synchronization mechanisms, such as locks, semaphores, and distributed algorithms like distributed mutual exclusion.
- ❖ **Communication and Inter-Process Communication (IPC):** Effective coordination, data sharing, and message passing are integral to the functioning of MPD systems, necessitating the development of efficient communication protocols, inter-process communication mechanisms, and addressing concerns such as latency, bandwidth, and message delivery guarantees.
- ❖ **Load Balancing and Resource Management:** Effectively distributing the workload among multiple processors means like applying task distribution, resource heterogeneity and adapting changes. Additionally, proficient resource management techniques are necessary to handle resources like memory, I/O devices, and network bandwidth across distributed nodes.
- ❖ **Fault Tolerance and Reliability:** MPD systems are prone to failures, including processor failures, node failures, or network failures. Techniques like replication, checkpointing, process migration, and distributed consensus protocols are employed to ensure system reliability and availability.

- ❖ **Security and Privacy:** Protecting sensitive data and ensuring secure communication in MPD systems pose significant challenges. Designing and implementing robust security measures, including authentication, access control, encryption, and intrusion detection systems, become critical in distributed environments.
- ❖ **Debugging and Performance Analysis:** Identifying and diagnosing issues, monitoring performance metrics, and analyzing system behavior across multiple processors.
- ❖ **Heterogeneity and Diversity:** MPD systems often involve processors or nodes with different architectures, capabilities, or operating systems. Managing heterogeneity, including different instruction sets, memory models, and communication protocols, adds complexity to the design and development process.

9 Write short notes on:

- ❖ **Inter-process Communication in a typical MPD OS:** refers to the mechanisms and techniques used for communication and data exchange between different processes or tasks running on multiple processors or nodes. Common IPC mechanisms include shared memory, message passing, pipes, sockets, and remote procedure calls (RPCs). These mechanisms facilitate communication and synchronization among processes, allowing them to exchange data, request services, and coordinate their activities.
- ❖ **Availability of resources in MPD systems:** refers to the accessibility and readiness of computing resources for processing tasks such as processors, memory, I/O devices, network bandwidth, and other system components.
- ❖ **Fault Tolerance in MPD systems:** involves designing and implementing mechanisms to handle failures and ensure system reliability and availability. Faults include processor failures, node failures, communication failures, and software errors. Fault tolerance techniques include redundancy, replication, error detection, error recovery, and fault-tolerant algorithms. These techniques aim to detect failures, mask their impact, recover from failures, and maintain system functionality even in the presence of faults, thereby enhancing the system's resilience.
- ❖ **Logical Clock:** refers to a mechanism used to order events and establish a partial ordering of events in a distributed environment where there is no global clock. Logical clocks provide a logical time concept that allows processes or nodes to agree on the ordering of events, even if they do not have access to a common physical clock such as Lamport's logical clocks or vector clocks, which assign timestamps or vectors to events and enable the determination of causal relationships among events in a distributed system.
- ❖ **Mutual Exclusion:** refers to a concurrency control mechanism that ensures that only one process or thread can access a shared resource at any given time. Multiple processes or nodes may compete for shared resources, mutual exclusion is crucial to prevent race conditions and maintain data integrity. Techniques such as locks, semaphores, and critical sections are used to implement mutual exclusion. These allow processes to acquire exclusive access to shared resources, ensuring that conflicting operations do not occur simultaneously.
- ❖ **Distributed File System:** is a file system that spans multiple nodes or machines in a distributed system that offer features such as file sharing, replication, fault tolerance, scalability, and data consistency. Examples include NFS (Network File System), HDFS (Hadoop Distributed File System), and Ceph. These file systems employ various techniques like file partitioning, data replication, caching, and distributed metadata management to provide efficient and reliable file access and management in distributed environments.

REFERENCES

- https://www.ijsr.net/archive/v4i4/SUB153187.pdf
- https://teachcomputerscience.com/distributed-operating-system/#:~:text=Architecture%20of%20a%20Distributed%20Operating%20System%3A,-In%20a%20DOS&text=All%20software%20and%20hardware%20compounds,and%20clients%20use%20these%20resources.
- https://docs.kernel.org/scheduler/sched-energy.html#:~:text=Energy%20Aware%20Scheduling%20(or%20EAS,the%20energy%20consumed%20by%20CPUs.
- https://www.javatpoint.com/microkernel-vs-monolithic-kernel
- https://www.theengineeringprojects.com/2021/06/real-time-embedded-systems-definition-types-examples-and-applications.html#:~:text=Following%20are%20the%20characteristics%20of%20firm%20real%20time%20embedded%20systems,and%20discard%20the%20delayed%20response.