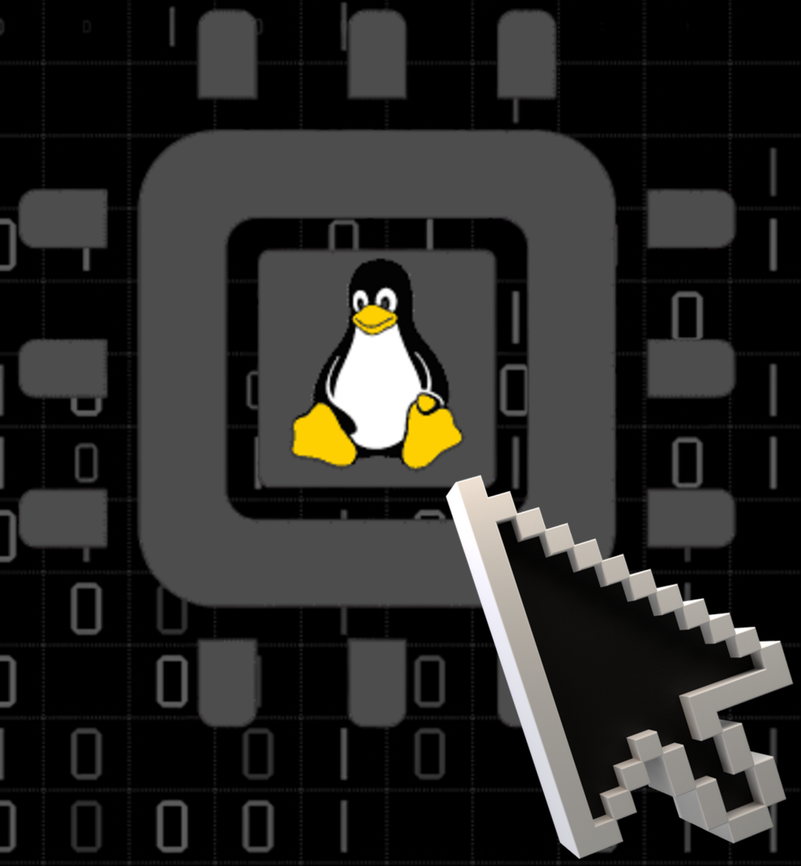
Embedded OS

OPERATING SYSTEM
PRESENTATION



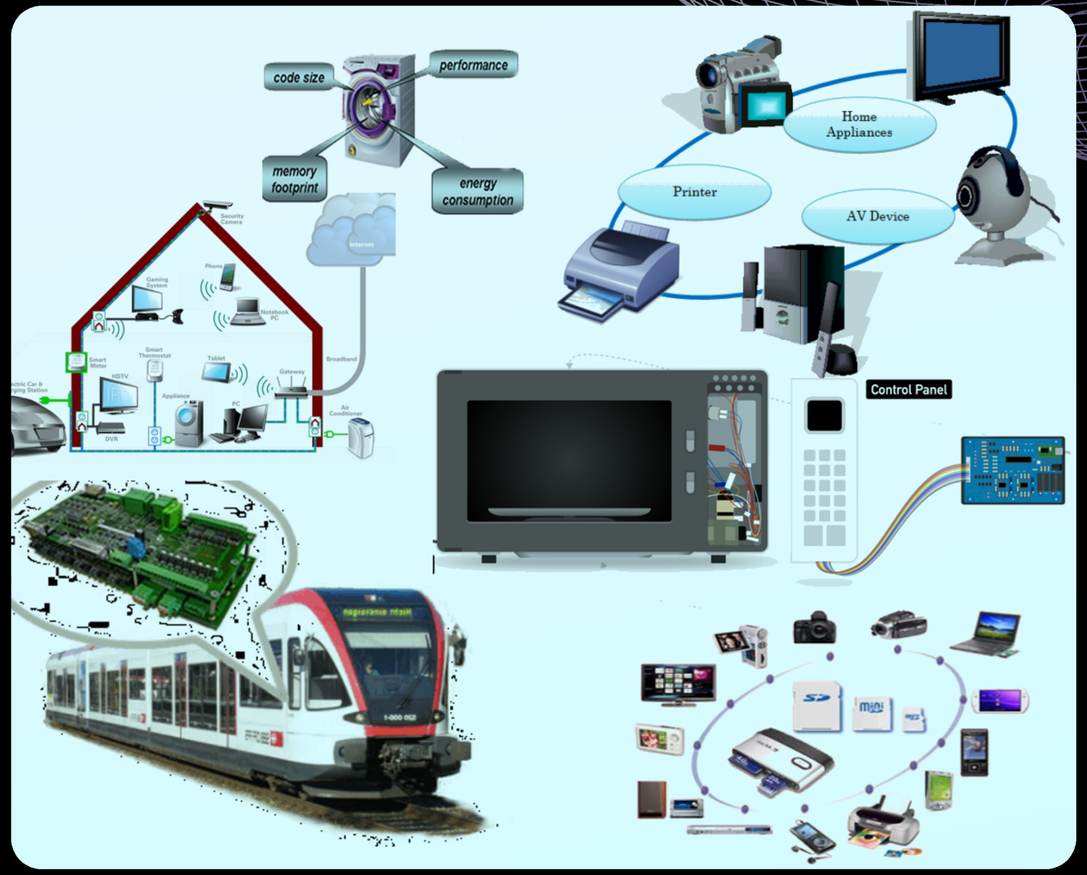
PRESENTED BY:

BISHAL GIRI
AAYAM TIWARI
ANUSHKA ARYAL
KUMUD DHUNGANA
PARAKRAM POUDEL

What is an Embedded System?

An embedded system is a small, specialized computer built into a larger device to perform a specific task — efficiently, reliably, and often in real time

Think of a smart washing machine: It uses an embedded system to sense water level, control the motor, manage cycles — all automatically, smart bulb, thermostat, modern rockets, microwave, etc



What is an Embedded OS?

An OS designed to operate in embedded systems

Cool Real-World Example: NASA Mars Rover
NASA’s Perseverance Rover runs on an embedded OScalledVxWorks. It enables the rover to:

- React to terrain in real time.Manage multiple sensors and robotic arms.Operate independently, millions of kilometers from Earth.
- A powerful OS in a compact system — exploring Mars

Key Features

1. Real-time responsiveness

The system can respond to inputs or events within a guaranteed time frame, which is critical for time-sensitive applications.

2. Low memory footprint

Embedded OS are designed to operate with minimal memory usage, making them suitable for resource-constrained devices.

3. Efficient resource management

They optimize the use of CPU, memory, and peripherals to maintain high performance and stability.

4. Predictable behavior

The system exhibits consistent and deterministic responses, ensuring reliability in repetitive tasks.

Embedded OS vs General-purpose OS

Feature	Embedded OS	General OS
Size	Small	Large
Realtime	Yes	Not always
UI	Often headless	GUI-heavy
Updates	Rare/Manual	Frequent/Automatic

Real-Time Operating Systems (RTOS)

An RTOS is an operating system designed to deliver predictable timing and response, ensuring tasks execute within strict time constraints.

Types of RTOS:

Hard Real-Time Systems

- Timing is critical.
- Missing a deadline can cause system failure or loss of life/property.
- Example: Pacemaker, anti-lock braking system (ABS).

Firm Real-Time Systems

- Missing a deadline doesn't cause disaster but hurts performance.
- Some loss of quality is acceptable, but not ideal.
- Example: Online transaction systems, VoIP (Voice over IP).

Architecture of Embedded OS

This diagram represents the architecture of an embedded operating system. It shows how the kernel handles core functions like process, memory, and I/O management, while optional layers like middleware and device drivers provide additional functionality and hardware support.

Scheduler/ Process Management:

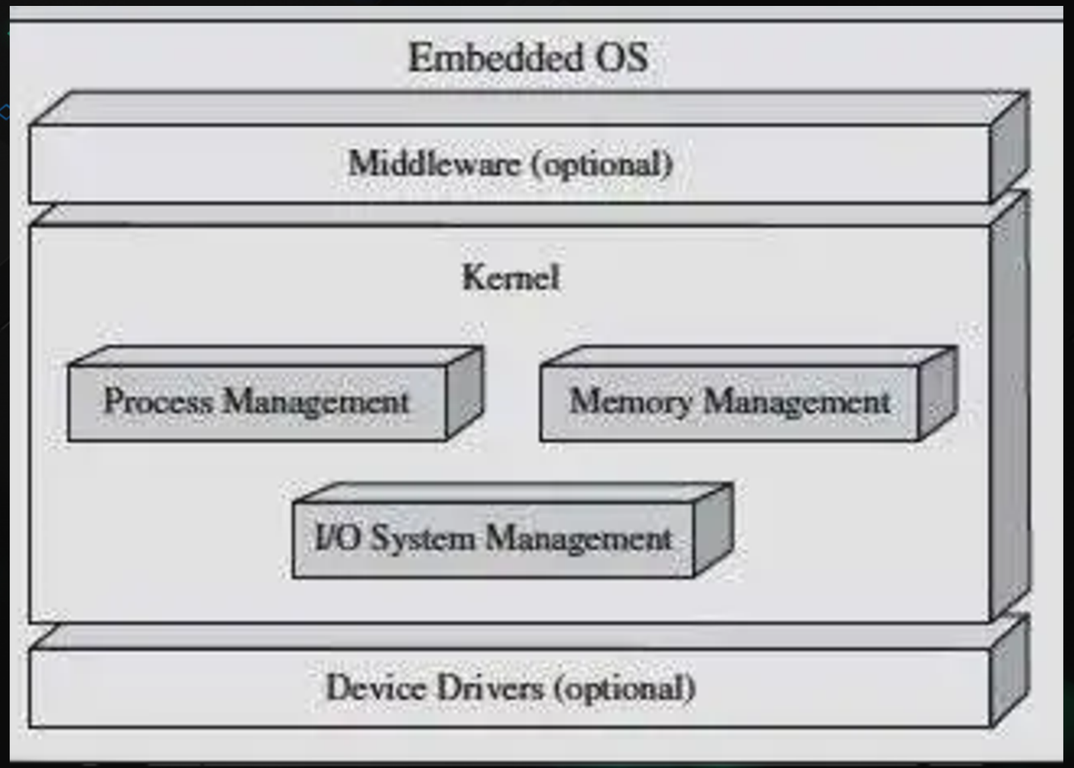
- Selects and manages the execution order of tasks based on priority and timing.

Memory Management

- Allocates and controls memory usage to ensure safe and efficient operation

I/O System Management:

- Manages communication between the CPU and hardware devices through drivers and interrupts.



Kernel and it’sTypes

The kernel is the central part of an OS that controls hardware and manages core tasks like memory, processes, and I/O.

Types of Kernels:

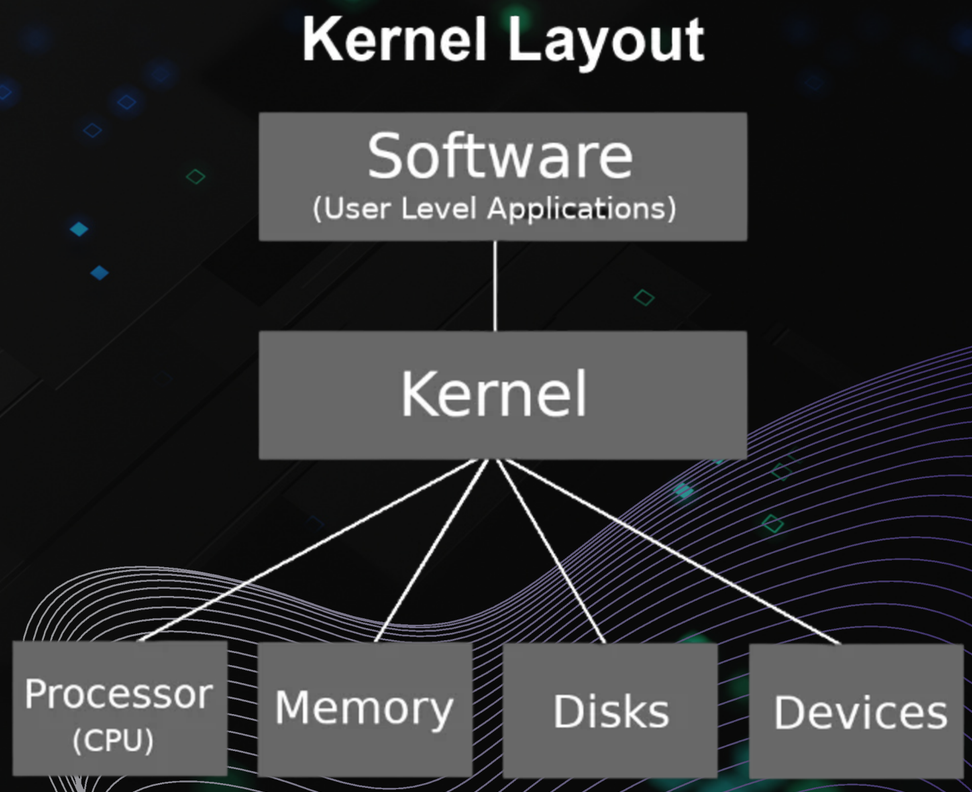
1. Monolithic Kernel
2. Microkernel

Monolithic Kernel

- All OS services run in kernel space
- Fast performance, but large and less modular
- Examples: Linux, VxWorks

Microkernel

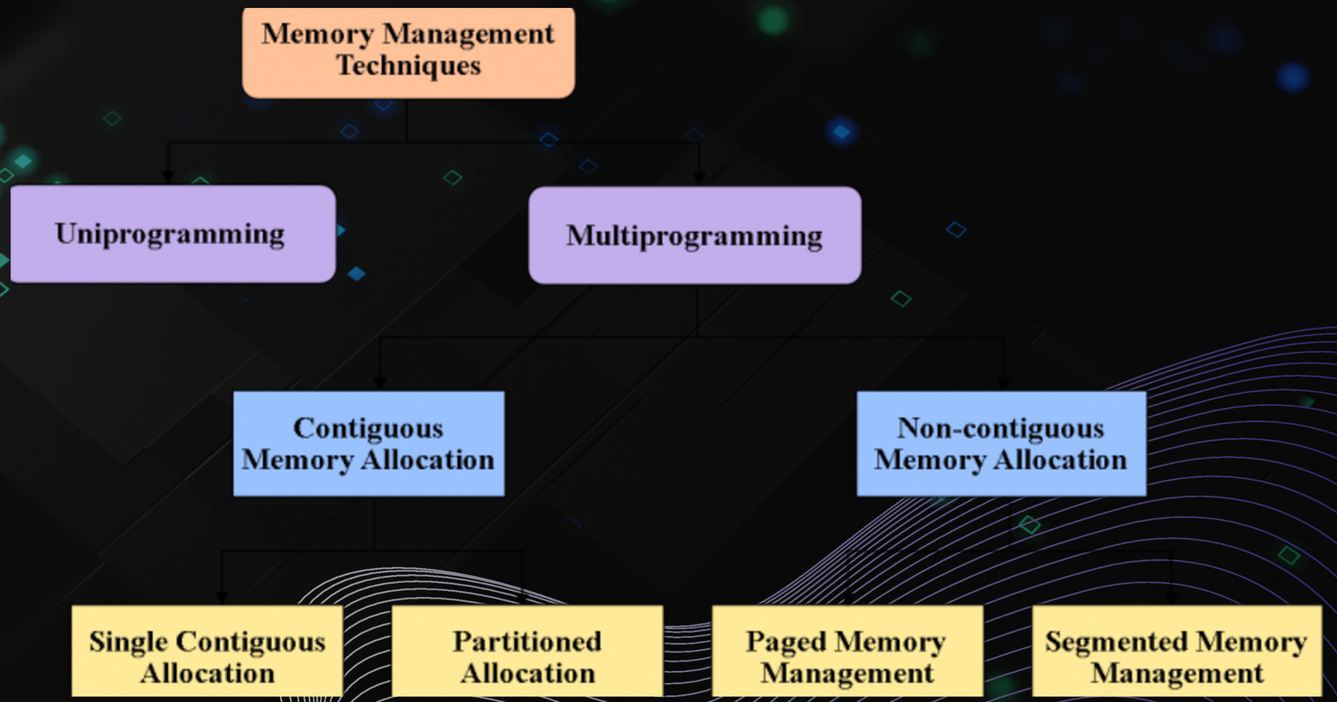
- Only core functions in kernel (e.g., scheduling, IPC)
- Other services (e.g., drivers, file systems) in user space
- More modular and secure, but slightly slower
- Examples: QNX, MINIX, L4



Memory Mangement in Embedded systems

Memory management in embedded systems is critical due to the following reasons:

- Limited resources: Embedded devices often have constrained memory (RAM and storage) because they are designed to be compact and energy-efficient.
- Real-time requirements: Many embedded systems are real-time systems, where meeting deadlines is crucial for system functionality.
- Reliability and efficiency: Since embedded systems often run for long periods without manual intervention, efficient memory use is necessary to ensure stability and prevent failures like crashes or slowdowns.



Examples of Embedded Operating Systems

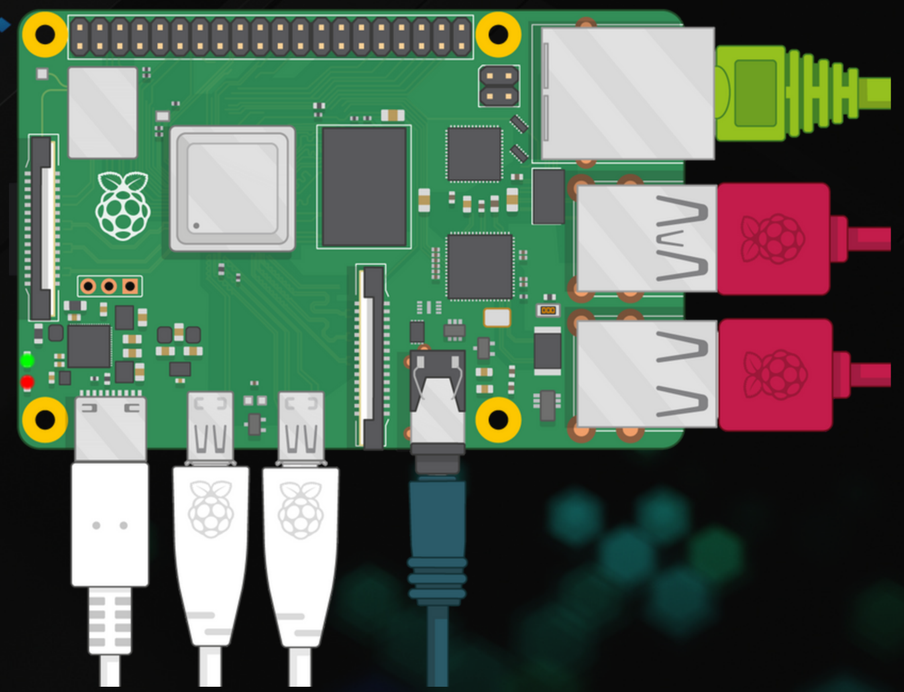
1. **FreeRTOS**

- FreeRTOS is an open-source, real-time operating system (RTOS) designed for embedded systems, particularly those with limited resources such as microcontrollers and small processors.
- Ideal for microcontrollers and small embedded systems: This makes it an excellent choice for devices with limited processing power and memory.
- Used in: Wearables, IoT sensors, smart appliances, and more.
- It’s commonly found in devices that require low power consumption, small memory, and real-time performance.

2. Embedded Linux

Embedded Linux is a highly customizable operating system specifically tailored for embedded systems. It takes the core functionality of the Linux kernel and adapts it to meet the unique requirements of devices with limited resources

- Used in:Raspberry Pi: A low-cost platform for DIY projects and educational purposes.
- Smart TVs: Provides a rich multimedia experience and networking capabilities.
- Networking Devices (e.g., Routers, Gateways): Reliable, scalable, and secure connectivity.



Key Features:

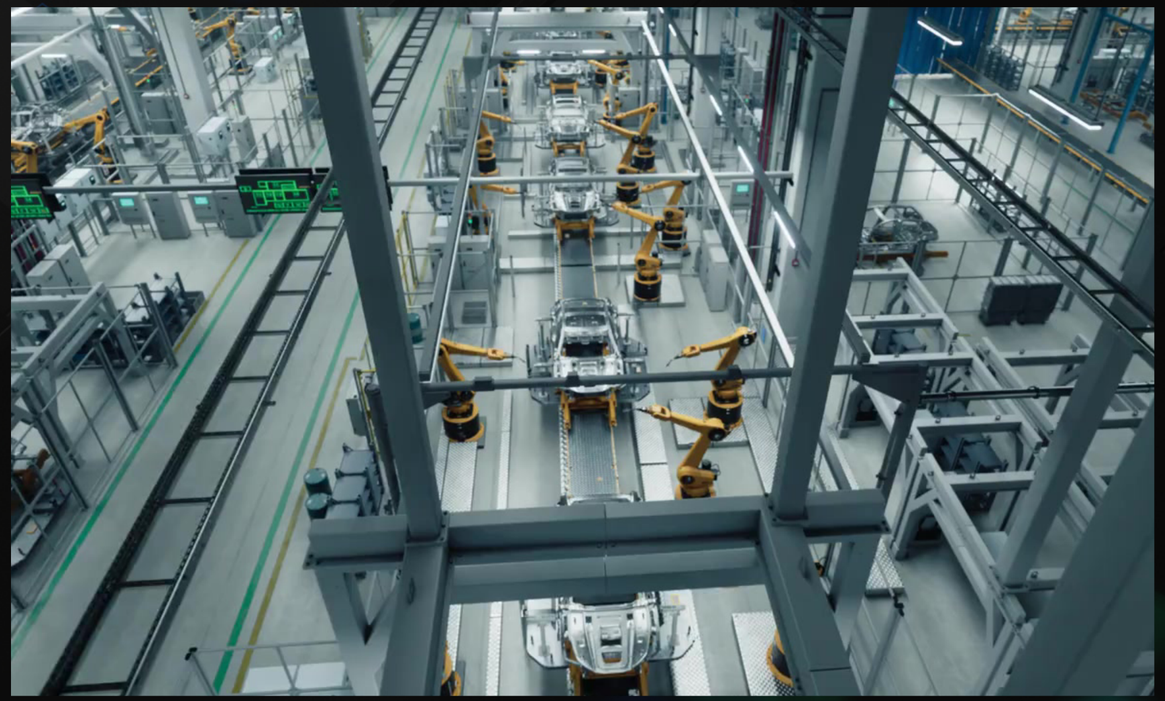
- Scalability: Can be used on everything from small microcontrollers to powerful embedded systems.
- Comprehensive driver support: Supports a wide range of hardware, from sensors to sophisticated peripherals.

3. VxWorks

VxWorks is a real-time operating system (RTOS) developed by Wind River, designed for embedded systems that require high reliability, real-time performance, and safety-critical capabilities

A Real-Time Operating System (RTOS)

- VxWorks is a commercial RTOS widely used in mission-critical and safety-critical embedded systems that require high reliability and deterministic performance.



Used in:

- Aerospace & Defense: VxWorks is commonly used in spacecraft, military systems, and satellites, where timing, reliability, and performance are paramount.
- Industrial Automation: For controlling robotics, manufacturing systems, and other automation tools.
- Medical Devices: In devices requiring precise control and timing, such as infusion pumps, patient monitors, and diagnostic equipment.

Challenges & Future Trends

Security

- Need for secure firmware updates, encryption, and tamper resistance
- Limit access to protect against hacking in critical systems
- “A hacked microwave or pacemaker isn’t just annoying — it’s dangerous”

Scalability

- Systems must scale from tiny devices (sensors) to complex platforms (robots)
- Reuse of code, modularity, and upgrade paths are becoming crucial

AI/ML Integration at the Edge

- AI models (especially pre-trained models) now run on embedded devices
- Enables local decision-making (e.g., detecting gestures or faults)

Conclusion

Embedded OS is essential in modern smart devices

Balances performance, reliability, and power

Key enabler for IoT, automation, and smart tech

Continues to evolve with AI and edge computing integration

References

- https://www.tutorialspoint.com/what-is-an-embedded-operating-system
- https://www.freertos.org/Documentation/00-Overview
- https://blog.felgo.com/embedded/embedded-operating-systems
- https://www.geeksforgeeks.org/computer-organization-architecture/introduction-of-embedded-systems-set-1/

Thank you!

Any Questions?