Comparison of IoT Operating Systems

Recep Tayyip Erdogan University
Computer Engineering Department
Internet of Things Lecture

Why Have an OS?

- Resource constraints (low CPU, memory, power)
- Abstraction layer for hardware and software
- Isolation between hardware, software, and network

What Does an OS Provide?

- Drivers
- Network stack
- HAL (Hardware Abstraction Layer)

Network Stack Structure

- Link layer (e.g., Bluetooth, Wi-Fi, 802.15.4)
- Network layer (IPv4, IPv6, 6LoWPAN)
- Application layer (CoAP, HTTP)

Hardware Abstraction Layer (HAL)

- Bridges hardware and software
- Standardizes interfaces
- Enables device independence

Device Independence

- OS isolates hardware from software
- No direct interaction between hardware and software

OS Services

- Timers, threads, queues
- Interrupt handling
- Resource management

Examples of IoT Operating Systems

- Zephyr: Open-source, supports many boards
- Mbed: For ARM-based systems, RTOS optional
- Riot-OS: Community-maintained, low memory footprint
- Contiki-NG: Focus on ARM and MSP430
- FreeRTOS: Used by Google's Thread
- Ubuntu Core: More secure, reliable
- Linux/Arduino (not a full OS)

Zephyr OS

- Open-source
- Supports many boards
- Features: Bluetooth Low Energy, Wi-Fi, 802.15.4, 6LoWPAN, CoAP

Mbed OS

- For ARM-based systems
- Optional RTOS
- Features: 6LoWPAN based, Apache 2.0 open source

Riot-OS Overview

- Open-source community maintained
- Low memory footprint (3kB RAM/flash)
- Preemptive multi-threading
- Deterministic scheduling

RIOT Structure

- Microkernel (3kB flash/ram)
- Internet standards support
- Development on Linux possible

Contiki-NG Overview

- Runs on fewer boards
- ARM and MSP430 support
- Cooja simulator for testing

Ubuntu Core

- More secure and reliable
- Minimal OS, easy on 1GB Raspberry Pi
- Uses snaps instead of Debian packages

Conclusion

- Key points: Abstraction, resource efficiency, device independence
- Importance of OS in IoT development

QA

- Any questions?