Introducción

El internet de las cosas (IoT) está transformando el mundo físico en una red de objetos interconectados. Detrás de esta revolución, los sistemas operativos juegan un rol fundamental al gestionar la comunicación, el hardware y la autonomía de los dispositivos. En este trabajo se exploran los conceptos clave del IoT, se analiza la clasificación y características de los sistemas operativos (SOs) que lo sustentan, y se profundiza en el caso de RIOT OS, un sistema operativo diseñado específicamente para entornos embebidos de bajo consumo.

¿Qué es el IoT?

Bahga y Madisetti (2015) definen el IoT como una infraestructura de red global y dinámica, que se configura automáticamente mediante protocolos de comunicación estándar. En este entorno, los objetos físicos y virtuales cuentan con identidades y utilizan interfaces, además, se integran sin problemas a la red de información, comunicando datos relevantes sobre los usuarios y su entorno.

Esta definición abarca de forma implícita los tres niveles arquitectónicos del IoT:

- (1) *Device layer* (capa de **dispositivos**).
- (2) *Communication layer* (capa de comunicación).
- (3) *Middleware and Application layer* (capa de servicios).

En este punto es necesario aclarar que el presente escrito se enfocará únicamente en los SOs utilizados en la capa de dispositivos. Esta capa, según Rajkumar Buyya y Amir V. Dastjerdi (2016), está compuesta por dispositivos con capacidades limitadas de procesamiento y comunicación. Típicamente alimentados por baterías, estos dispositivos están diseñados para recolectar datos del entorno, realizar un procesamiento local básico y transmitir los resultados hacia niveles superiores de la arquitectura.

Retomando la definición planteada por Bahga y Madisetti, de ésta se desprende una serie de características que, al ser analizadas, permiten comprender mejor el término IoT y, en consecuencia, la *device layer*.

Principales características del IoT

- *Es dinámica y autoajustable*: Los dispositivos y sistemas IoT tienen la capacidad de adaptarse de manera dinámica a los cambios en su contexto operativo, así como de tomar acciones en respuesta a estos cambios. Por ejemplo, si se considera un sistema de vigilancia conformado por varias cámaras, entonces se tiene que las cámaras pueden adaptar sus modos (a normal o a infrarrojo) en función de si es día o de noche. Además, éstas pueden aumentar la resolución cuando se detecta algún movimiento y alertar a las otras cámaras para que hagan lo mismo.
- *Es autoconfigurable*: Los dispositivos IoT tienen la capacidad de configurarse a sí mismos. Esto hace referencia a que pueden inicializarse y conectarse automáticamente. Igualmente, implica que los objetos pueden actualizarse automáticamente. La importancia de esta característica radica en que los dispositivos puedan trabajar en conjunto para conseguir un objetivo común, pero de forma autónoma (minimizando la intervención humana).
- *Usa protocolos de comunicación estándar*: Los dispositivos admiten varios protocolos, lo que les permite comunicarse con distintos tipos de objetos.

- *Usa interfaces*: Las interfaces de los dispositivos IoT permiten a los usuarios consultarlos, supervisar su estado y controlarlos remotamente.
- *Cuenta con una identidad*: Cada uno de los dispositivos dentro de la infraestructura tiene un identificador único. Por ejemplo, una dirección IP.
- *Integrados en redes de información*: Los dispositivos forman parte de redes de información, puesto que éstos necesitan intercambiar información entre sí. En este sentido, los dispositivos pueden ser descubiertos de forma dinámica dentro de la red por otros objetos. En consecuencia, los dispositivos deben poder describirse a sí mismos a los demás elementos.

Como se puede observar, en todos los puntos anteriores subyace un componente esencial: el sistema operativo. Éste permite que los **dispositivos** cumplan las características listadas. En este sentido, estos objetos necesitan un sistema operativo, por ejemplo, para manejar sensores y actuadores, controlar el uso de batería y recursos, coordinar el trabajo cooperativo en red y gestionar la conexión a ésta, administrar la descarga e instalación de actualizaciones, manejar protocolos de comunicación y descubrimiento automático, asignar o recibir y proteger la IP, proporcionar mecanismos de autodescripción, así como permitir la comunicación con otras capas del sistema.

Sistemas Operativos

Un SO, según Remzi y Andrea Arpaci Dusseau (2014), es un pedazo de software que se encarga de facilitar la ejecución de programas (permitiendo inclusive ejecutar varios simultáneamente), permitiéndoles compartir memoria e interactuar con otros dispositivos. Este programa, que debe ser fácil de usar, se asegura de que la computadora opere eficientemente.

En la actualidad, cuando se habla de los sistemas operativos, generalmente, se piensa en Windows, macOS, Android, entre otros. Esto se debe a que están presentes en dispositivos de uso cotidiano (laptops y celulares). Sin embargo, como se ha visto, existen SOs especializados en equipos más simples y, por ello, menos conocidos.

Principales características de los diferentes tipos de SOs

Característica	SOs tradicionales	SOs para *Device Layer*
Dispositivos objetivo	PC, laptops, servidores, celulares	Sensores, microcontroladores, actuadores
Ejemplos comunes	Windows, mac, Android	FreeRTOS, Zephyr, Mbed, RIOT, Contiki, Tiny
Recursos disponibles	CPU potente, mucha RAM y almacenamiento	CPU limitada, poca RAM y memoria flash
Multitarea	Completa (con planificación avanzada)	Simplificada o cooperativa (según el caso)
Interfaz de usuario	Gráfica (GUI) o CLI	Generalmente no tiene o es muy básica
Gestión de energía	Sí, aunque menos crítica que en dispositivos embebidos	Crítica (bajo consumo)

Tamaño del SO	Grande (GB o varios cientos de MB)	Muy pequeño (KB o pocos MB)
Tiempo real	No necesariamente	Sí, en la mayoría de los casos
Actualizaciones y mantenimiento	Frecuentes, complejas	Más controladas, a veces remotas (OTA)
Costo de implementación	Libre o incluido en el costo del hardware; algunos con licencias propietarias	Bajo o gratuito (muchos son open-source)

Se han visto ya las diferencias entre los SOs tradicionales y los enfocados en los **dispositivos**. Sin embargo, estos últimos presentan, igualmente, diferencias significativas entre sí.

Clasificación de SOs para *Device Layer*

- *Por tipo de respuesta: RTOS vs GPOS*. Los sistemas operativos de tiempo real (Real-time operating system) están diseñados para responder a eventos dentro de un tiempo predecible. Esto significa que el sistema siempre responde en el mismo tiempo o dentro de un margen muy pequeño. En este sentido, lo que los distingue de los SOs de propósito general (General Purpose operating system) es el tiempo de respuesta garantizado que los hace ideales para sistemas embebidos o de misión crítica.
  Para lograr esto, generalmente, los RTOS son ligeros y su planificación multitarea es determinista. Por su parte, los GPOS son más pesados y su tiempo de respuesta puede variar. Además, suelen ofrecer otras funcionalidades, como gráficos y sistemas de archivos complejos. Son utilizados en los **dispositivos** en raras ocasiones.

- *Por arquitectura: event-driven vs multithreaded*. Los SOs basados en eventos (event-driven) están estructurados como un bucle principal que responde, precisamente, a eventos cuando ocurren (interruptores, timers, mensajes, etc.). Sus principales características son las siguientes: (1) tiene un solo hilo principal (sin concurrencia) y (2) reacciona a los eventos con *handlers* (funciones cortas y rápidas).
  Por su parte, los SOs multihilo (multithreaded) tienen varios hilos de ejecución, y un planificador decide cuál se ejecuta en cada momento. Puede tener planificación preemptiva (interrupciones entre tareas) o cooperativa (ceden el control voluntariamente).

Ejemplos de SOs para *Device Layer*

- *Tiny*. Es un sistema operativo con licencia BSD, basado en una arquitectura orientada a eventos y no está diseñado para aplicaciones en tiempo real. Se puede usar en *sensor boards* (placas de sensores) como la MTS400.

- *Contiki*. Este sistema operativo comparte arquitectura y licencia con *Tiny*, además, puede ser utilizado como RTOS, pero no es recomendable. Se puede usar en microcontroladores como el MSP430 y el NRF52840.

- *RIOT*. Es un sistema operativo en tiempo real, multihilo y distribuido bajo la licencia LGPL versión 2.1. Se puede usar en una raspberry pi pico y en una raspberry pi pico 2 w.

Entre los sistemas revisados, RIOT OS destaca por su alta compatibilidad y desarrollo activo, por lo tanto, se hará una exploración más profunda de éste. A continuación, se describe brevemente su filosofía de diseño y se implementa una pequeña demostración de red para observar su funcionamiento práctico entre dos nodos.

RIOT OS

¿Qué es?

RIOT es un sistema operativo libre y de código abierto desarrollado por una comunidad diversa que incluye empresas, instituciones académicas y entusiastas. Su objetivo es implementar estándares abiertos relevantes para un IoT conectado, seguro, duradero y respetuoso de la privacidad. (RIOT Website, 2025)

En una aplicación básica, este SO puede funcionar de forma estable con aproximadamente 1.5 KB de RAM y 5 KB de almacenamiento.

Ejemplo de uso

Se utilizará una computadora Asus Vivobook con 16 GB de RAM, 512 GB de almacenamiento y un procesador Ryzen 5 5600H. En ésta se instalará y utilizará RIOT OS siguiendo las instrucciones que se muestran a continuación:

```
git clone https://github.com/RIOT-OS/RIOT.git

# Crear taps
sudo ip tuntap add mode tap tap0
sudo ip tuntap add mode tap tap1

# Crear bridge
sudo ip link add name br0 type bridge

# Añadir interfaces TAP al bridge
sudo ip link set tap0 master br0
sudo ip link set tap1 master br0

# Activar interfaces
sudo ip link set br0 up
sudo ip link set tap0 up
sudo ip link set tap1 up

# En este punto es necesario abrir dos terminales y ejecutar en ambas la
siguiente instruccion

# Directorio de trabajo
cd RIOT/examples/networking/gnrc/gnrc_networking

# Terminal 1
PORT=tap0 make -C . BOARD=native
sudo ./bin/native64/gnrc_networking.elf tap0
```

```
# Terminal 2
PORT=tap1 make -C . BOARD=native
sudo ./bin/native64/gnrc_networking.elf tap1

# Terminal 1
ifconfig

# Terminal 2
ping <inet6 addr Terminal 1>
```

Este procedimiento permitió ver cómo RIOT sirve para comunicar dos dispositivos IoT, los cuales fueron simulados mediante dos procesos de la computadora.

Conclusiones

A lo largo de este trabajo se exploró el papel fundamental que juegan los sistemas operativos dentro del ecosistema del Internet de las Cosas, particularmente en la *device layer*, donde los recursos son limitados y las exigencias de autonomía son elevadas. Se analizaron las características clave de esta capa y se compararon distintos tipos de sistemas operativos, destacando las diferencias entre los tradicionales y los diseñados para dispositivos embebidos.

Referencias

- *IoT - A hands-on approach*. Recuperado de https://jcer.in/jcer-docs/E-Learning/Digital%20Library%20/E-Books/Internet-of-things-a-hands-on-approach-%20Arshadeep.pdf
- *Operating Systems: Three Easy Pieces*. Recuperado de https://archive.org/details/operating-systems-three-easy-pieces/page/34/mode/2up
- *IoT - Principles and Paradigms*. Recuperado de http://dhoto.lecturer.pens.ac.id/lecture_notes/internet_of_things/Internet%20of%20Things%20Principles%20and%20Paradigms.pdf
- Iot Os: Examples. Recuperado de https://www.geeksforgeeks.org/iot-operating-systems/
- Clasificación de los OS de IoT - https://inria.hal.science/hal-00945122/PDF/2013-riot_os.pdf
- [1] RIOT OS. https://doc.riot-os.org/index.html#the-quickest-start
- RTOS definition. https://www.ibm.com/think/topics/real-time-operating-system
- Tiny OS. http://www.tinyos.net/
- Contiki OS. https://github.com/contiki-ng/contiki-ng/blob/develop/LICENSE.md
- SOs en el IoT. https://www.youtube.com/watch?v=Fx-iThPHQmg
- IoT OS. https://devopedia.org/iot-operating-systems