What's on the menu…

Software Comprehension and Maintenance
April 2005

printk("Topic=%s", "RF-IDs");

Achilleas Anagnostopoulos
Department of Management Science and Technology
Athens University Of Economics and Business

What are RF-IDs?

Radio frequency identification (**RFID**) is a method of remotely storing and retrieving data using devices called **RFID tags**. RFID tags contain antennas to enable them to receive and respond to radio-frequency queries from an RFID tranceiver. [Wikipedia]

RF-IDs are commonly used:

- Industry.
- Animal control.
- Security applications.

Expect to see RF-IDs soon in:

- Consumer products (clothes, cds…)
- Smart homes, digital canvas, personalized info kiosks.
- E-cash, toll-booths, patient monitoring.

RF-ID types

Active Tags

- Need power source.
- Higher cost but longer range and more memory.

Passive Tags

- Powered from reader.
- Cost is $0.40 => expected to drop to $0.05 by 2012.
- They are small! Range from **10mm to 5m.**

LF tags(125-134 Khz)

- Small Range.
- Commonly used for animal tracking.

HF tags(13.56 Mhz)

- book/pallet tracking
- ID badges/baggage tracking.

UHF tags(868-956Mhz)

- Pallet tracking.
- Vehicle tracking.

Microwave tags(2.45 Ghz)

- Long range access control for vehicles. (General Motors' OnStar system)

RF-IDs are neat! But how do we read/write them?

- Different tag types require different tranceivers.
- There are several commercial readers available.
- Each reader employs different comm. protocols.
- Each tranceiver requires a different connection interface. (Serial, Parallel, USB, PCI or ISA cards)
- Software developers just need to read/write tags without messing with the reader H/W itself.

SO, what we need is a standardized way for reading and writing tags regardless of the underlying reader H/W and communication protocols!

How can you implement a Hardware Abstraction Layer?

There are **two** schools of thought on this subject!

"That sounds interesting. Let's make an RF-ID library!"

- Libraries work! However there are extra dependencies for the developer.
- Unsatisfied developers will eventually roll-up their own libs. We will end up with *multiple* libs having *different interfaces* that developers need to support!

"That sounds interesting. Let's put that in the kernel!"

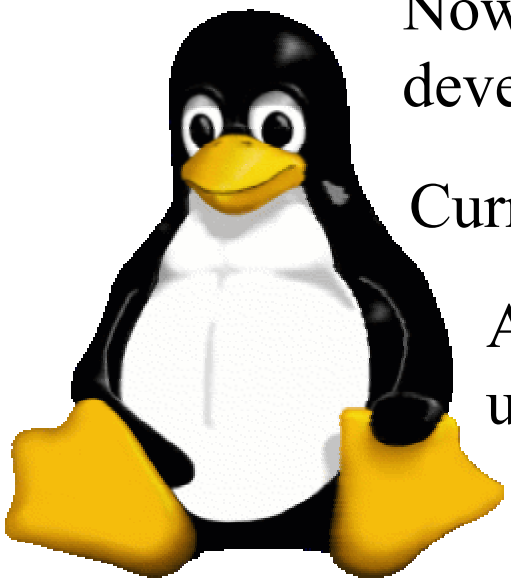
- We provide a common, documented interface to the underlying hardware.
- Developers may contribute code for supporting new readers; no need to change the HAL.
- May however lead to a code-bloated kernel(like windows)

The linux kernel OR what makes my favorite OS tick

Kernel: Low-level system software that provides a HAL, disk and filesystem control, multi-tasking, load-balancing, networking and security enforcement.

Kernel ≠ Operating System

The linux kernel was originally created by Linus Torvalds. Nowdays, it uses contributed code from thousand developers around the world. Released under the GPL.

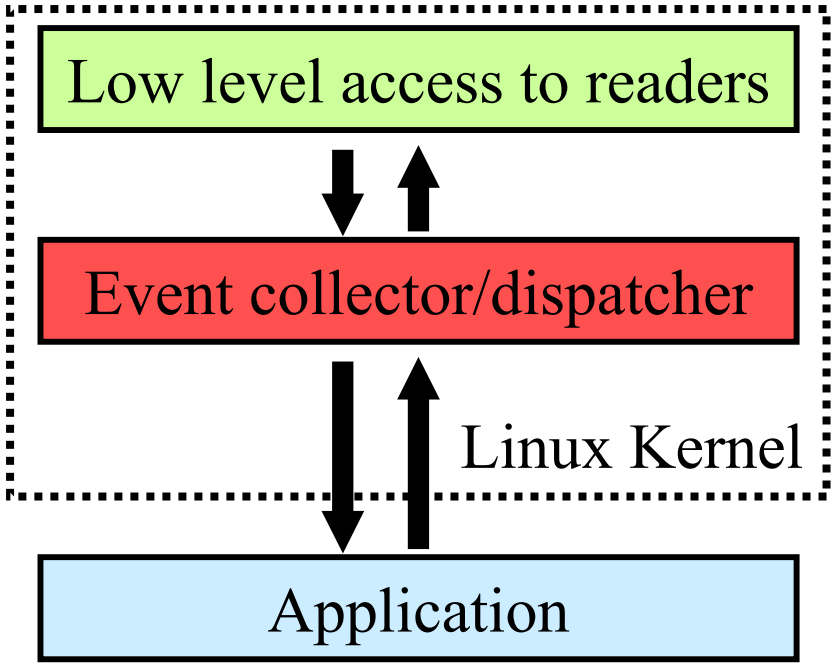


Current Version is **2.6.11.5** (available for d/l @ kernel.org)

A quick & dirty source count using the 'wc' tool tells us that the 2.6 kernel consists of:

4.4 MLOC in C and **248K** lines in Assembly

Adding an RF-ID service to the linux kernel



The low level part auto-detects and configures connected readers.

The event collector gathers events generated by readers and processes application commands.

Applications interface RF-ID tags in a generic way via the event collector.

That’s all…

Any Questions?