Embedded Open Source

Preview

- Hardware
- Market Trends
- Reasons for choosing Open Source
- Community Development
- Source Code
- Security
- Bug Tracking
- Documentation
- Tool chain

What is a device?

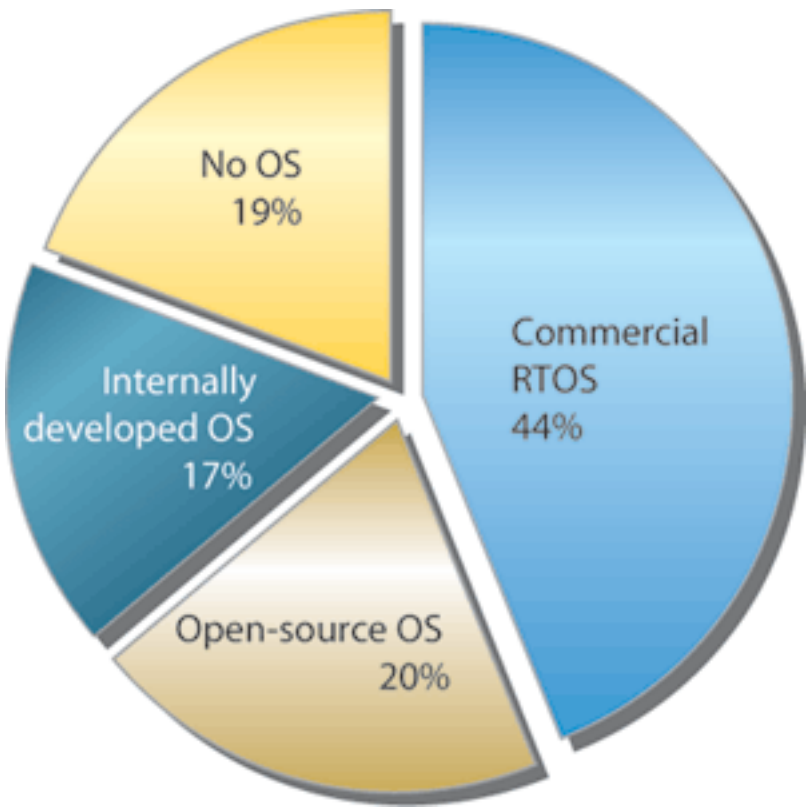
- May or may not have an operating system
- System built for a specific hardware expectation
- System typically built for a specific purpose

Embedded systems

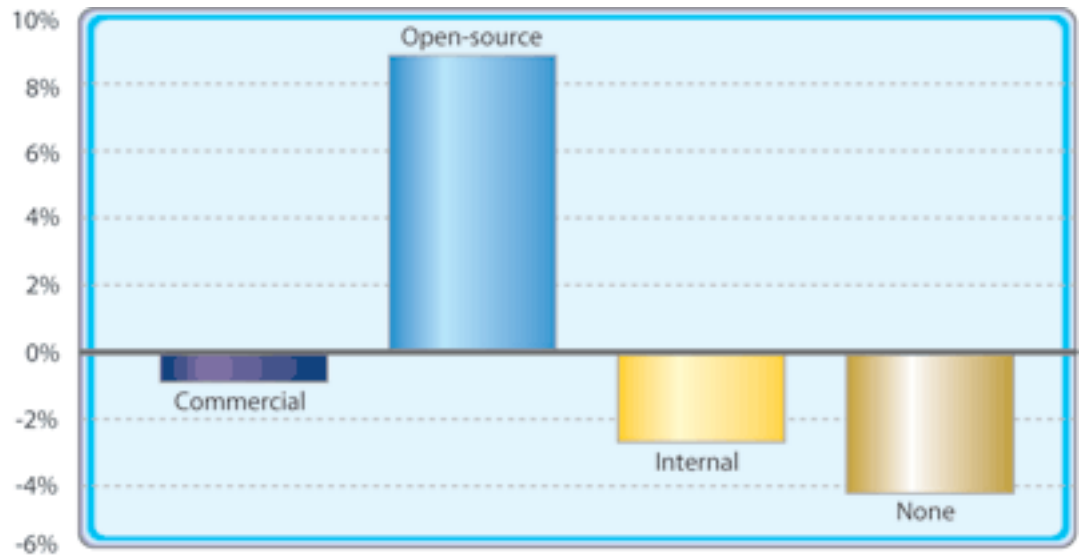
An embedded system is a special-purpose system in which the computer is completely encapsulated by the device it controls. Unlike a general-purpose computer, such as a personal computer, an embedded system performs one or a few pre-defined tasks, usually with very specific requirements. Since the system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product. Embedded systems are often mass-produced, so the cost savings may be multiplied by millions of items.

Source: wikipedia

Market Share



Current



Planned

Source: embedded.com survey

Reasons for Open Source

- Quality and reliability of code: modularity and structure, ease of fixing, extensibility, configurability, predictability, error recovery, longevity
- Availability of code
- Hardware support
- Communication protocol and software standards
- Available tools
- Community Support
- Licensing
- Vendor independence
- Cost: initial development setup, additional tools, runtime royalties.

Source: Building Embedded Linux Systems

Community Development

- Hardware
  - Developers need the hardware or simulator

Source Code

- Many eyes?
  - Fewer users
  - Fewer users see source code
  - Errors are harder to get information about
- Release early, release often?
  - Difficult to update systems makes frequent releases more difficult

Security

- Security updates may be difficult
- Physical security -- access to device
- Can’t easily replace hardware to solve a security issue
- Limit debug output in production?
- Read-only parts

Bugs

- Users report a “behavior”
- Difference between software and hardware bugs
- Bugs through all layers of the system
- Difficult to debug
  - May not always be able to get debug information out of device
- More than one solution: hw/sw

Documentation

- Need complete documentation for all layers
- User stories
- Keep with modules in revision control
- Document management system
- Keep track of hardware/software dependancies

Software Stack

- Bootloader
- Kernel
- Drivers
- Application

Bootloaders

- Difficult without BIOS to start executing instructions: init, test, stack pointers
- Specialized Software
  - JFlash-linux
- Tiny bootcode
  - Blob
  - Redboot
  - BootIdr

Kernel

- Linux
  - Real-time variants
- eCOS
- FreeRTOS
- TinyOS
- ChorusOS

Drivers

- Framebuffer
- I/O
- Filesystem
- Other

Application Layer

- Logically handles input and output
- User interface
- GUI
- Server

Tool chain

- Binutils
- Gcc
  - Cross compiling
- Glibc
  - Other “lite” versions of C libraries

Other tools

- Minicom
- In circuit emulators
  - JTAG emulators
- SSH
- Syslog