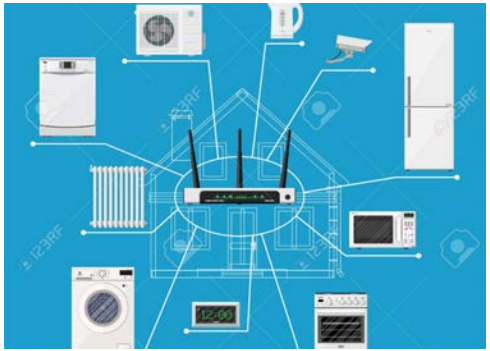
OPERATING SYSTEMS

EMBEDDED SYSTEMS

Embedded System

- The use of electronics and software within **a product that has a specific function or set of functions, as opposed to a general-purpose computer**

- smartphones, digital cameras, video cameras, calculators, home security systems, household appliances…



- …various automotive systems, and numerous types of sensors and actuators in automated systems

Embedded System

Embedded systems are often **tightly coupled to their environment**

Real-time constraints imposed by the need to interact with the environment

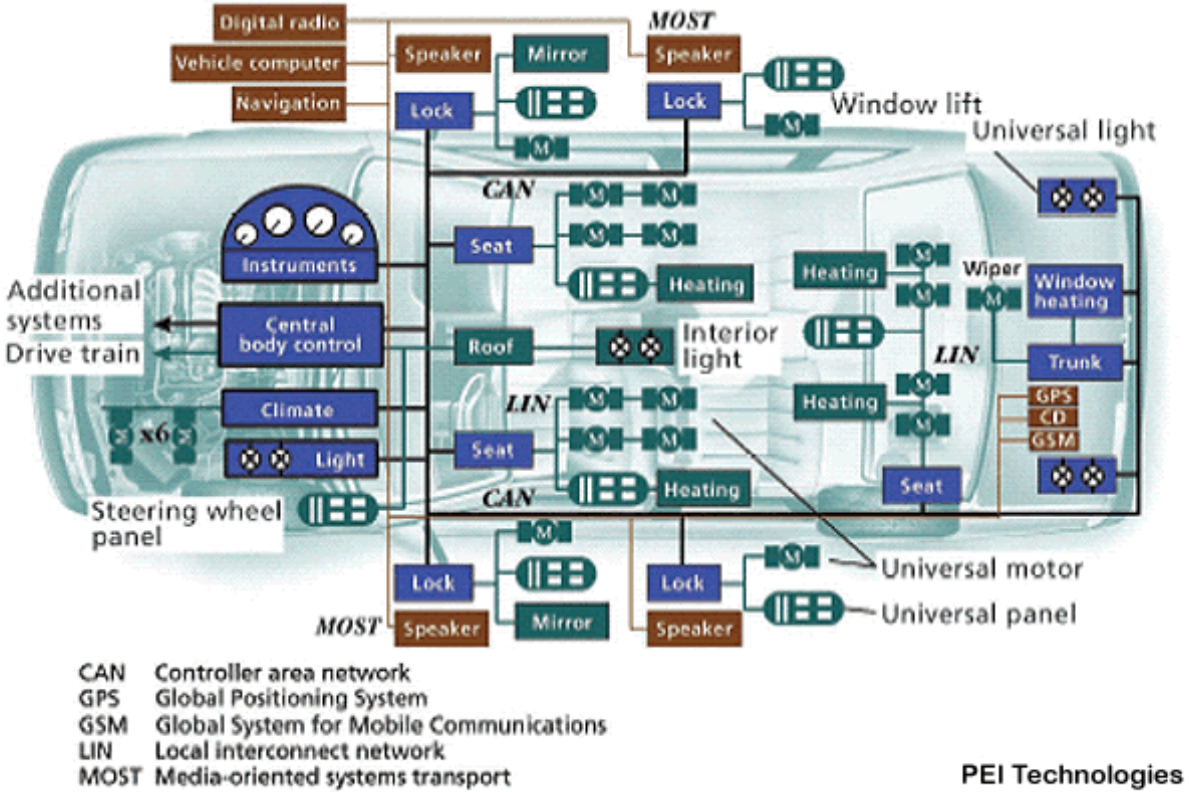
Constraints on speed, measurements, time durations, and the like, dictate **the timing of software operations**

If multiple activities must be managed simultaneously, this imposes more **complex real-time constraints**

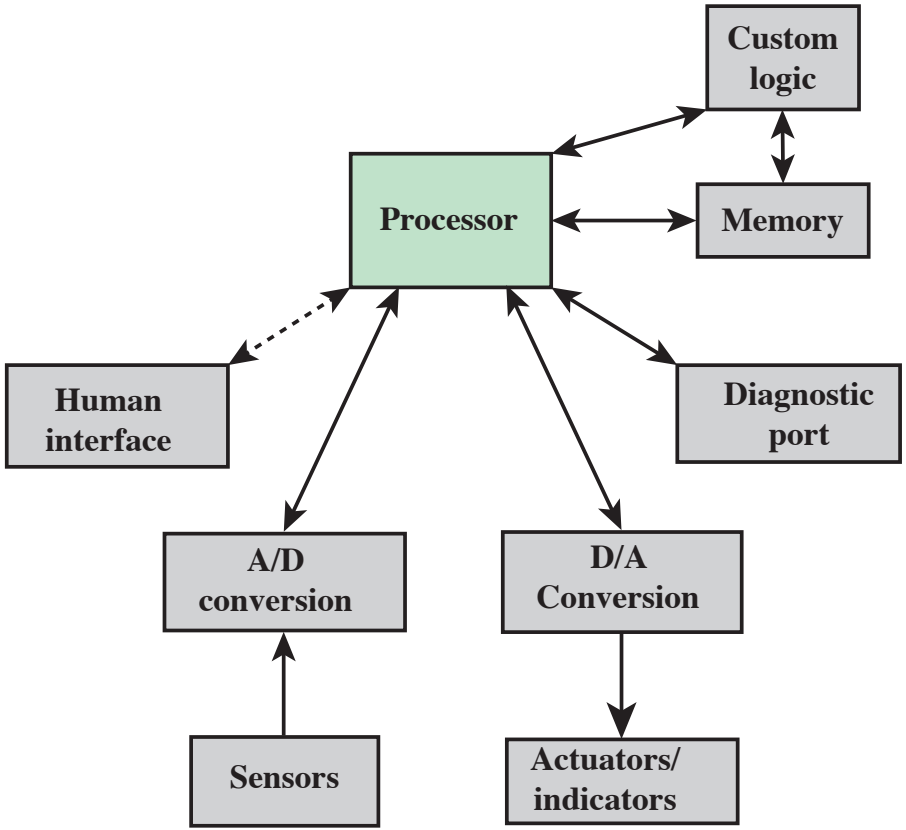
Application Processors vs Dedicated Processors

- Application Processor
  - General Purpose with ability to execute complex operating systems, such as Linux, Android, and Chrome
- Dedicated Processor
  - Specialized to perform one or a small number of specific tasks
  - The processor and associated components can be engineered to reduce size and cost
- An Embedded System is comprised by several dedicated processors and, optionally, one or more application processors

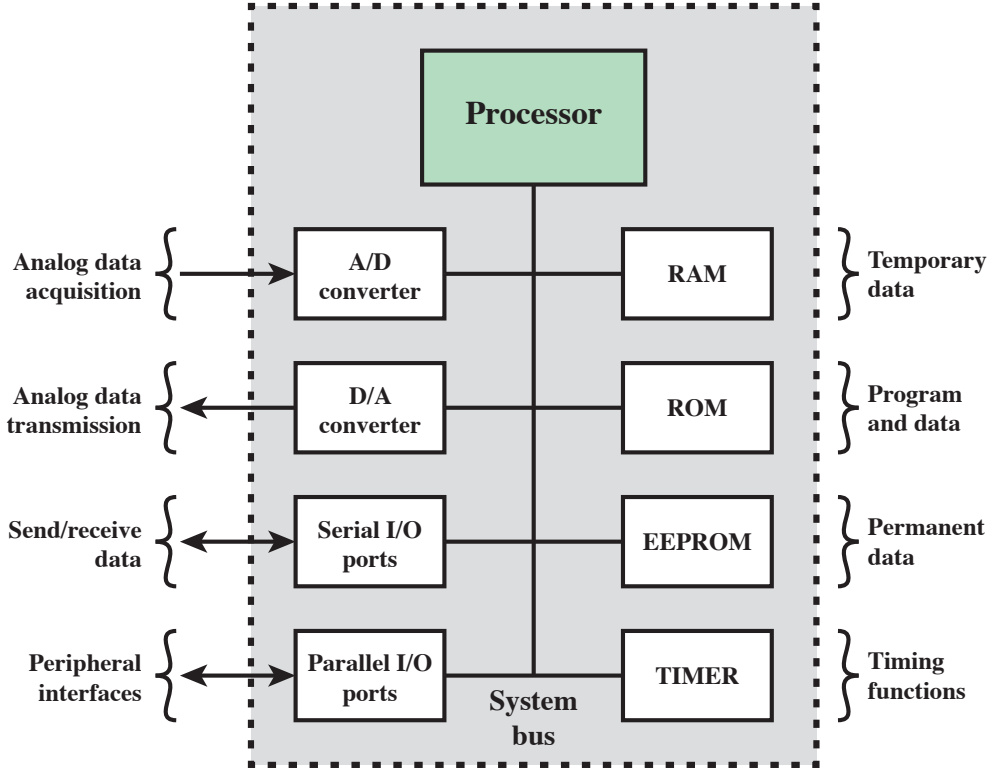
Automotive Systems



Organization of an embedded system



Typical microcontroller chip



Deeply Embedded System

- Relies on a microcontroller rather than on a microprocessor
  - It is not programmable once the program logic for the device has been burned into ROM
- No interaction with a user
- Dedicated, single-purpose devices that detect something in the environment, perform a basic level of processing, then do something with the results
- Extreme resource constraints in terms of memory, processor size, time, and power consumption
- The Internet of Things depends heavily on deeply embedded systems
  - Often wireless capability

Characteristics of Embedded OS

Real-time operation

Reactive operation

Configurability

I/O device flexibility

Streamlined protection mechanisms

Direct use of interrupts

Developing an Embedded OS

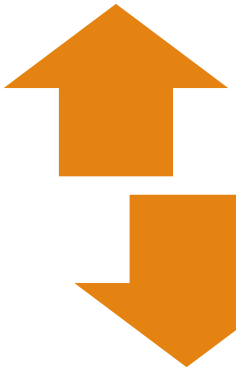
Two general approaches

- Take **an existing OS and adapt it** for the embedded application
- **Design and implement** an OS intended solely for embedded use

Adapting an existing OS

Adapting an Existing OS

- An existing commercial OS can be used for an embedded system by adding
  - Real time capability
  - Streamlining operation
  - Adding necessary functionality



Advantage

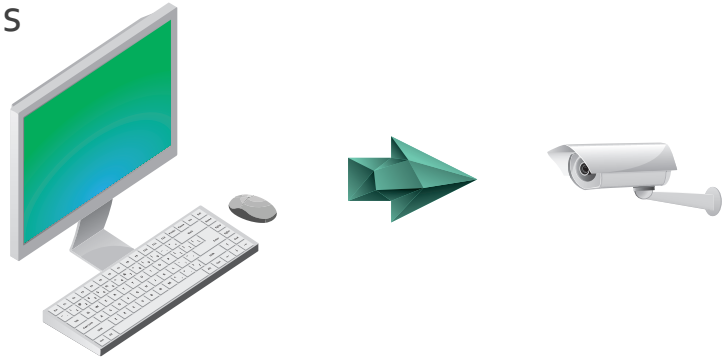
- Familiar interface

Disadvantage

- Not optimized for real-time and embedded applications

Cross Platform Development

- Typically, the development of an operating system is carried out on the same hardware platform it is built for.
- In the case of embedded system, development is carried out on a platform that is different from the target systems



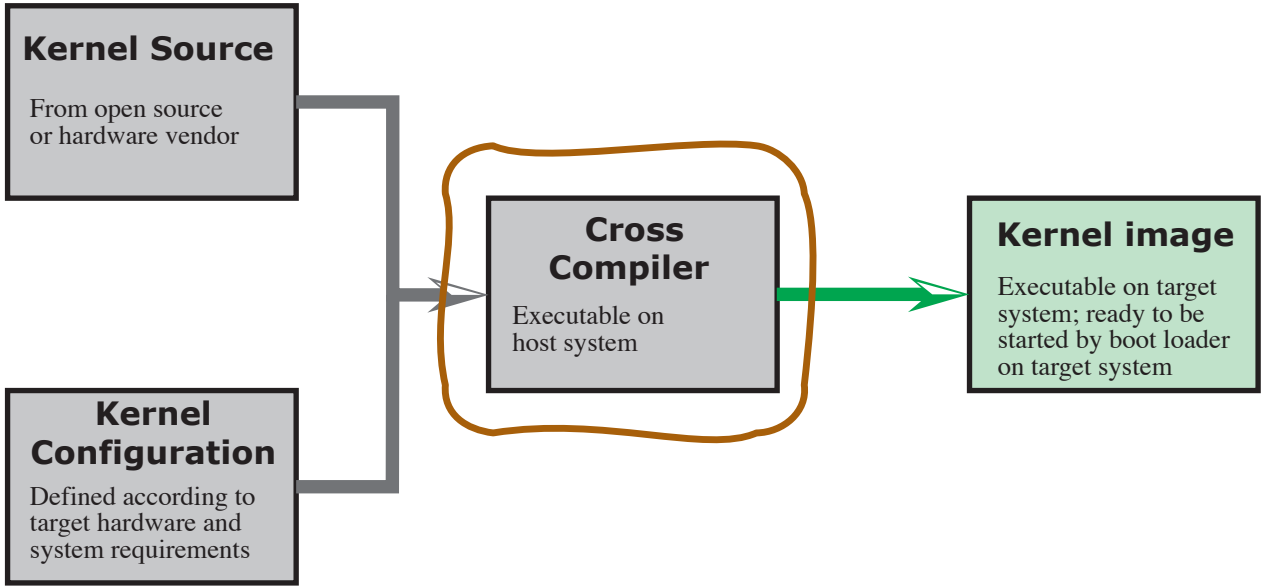
Host

- Cross-platform development environment

Target

- Kernel
- Root file system
- Boot loader

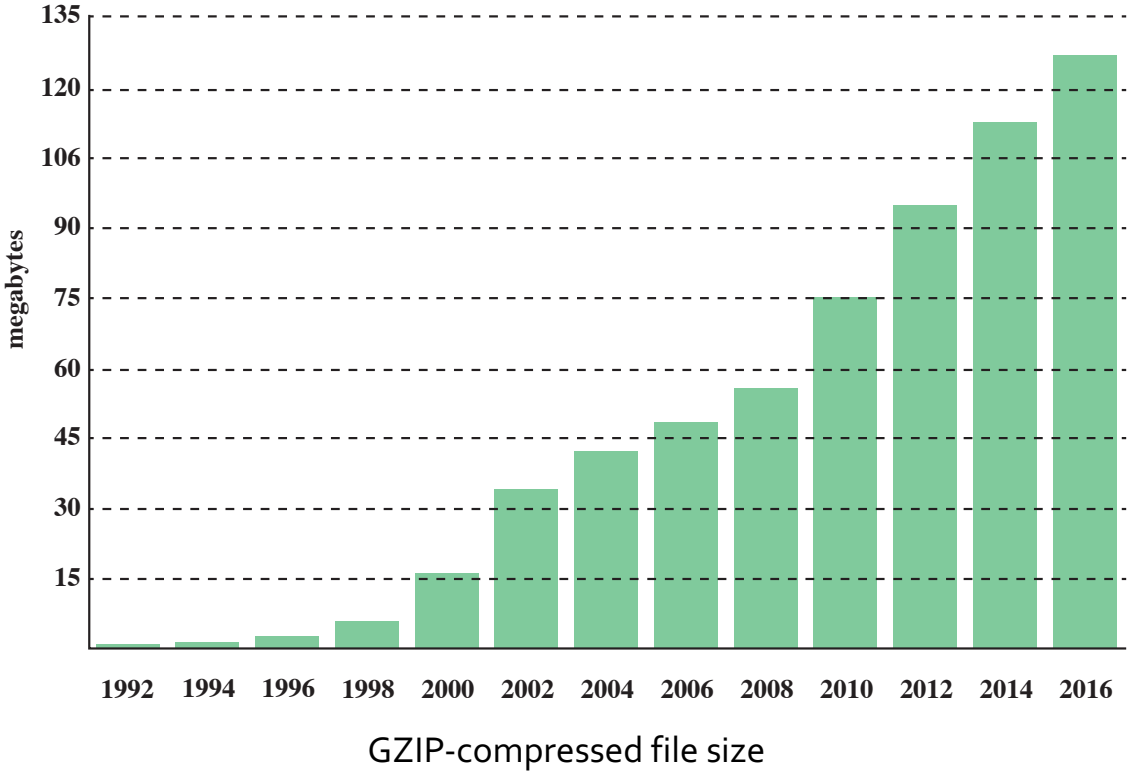
Kernel Compilation



Embedded Linux

- An embedded Linux distribution is **customized** for the size and hardware constraints of embedded devices
  - Includes software packages that support a variety of services and applications on those devices
  - **An embedded Linux kernel will be far smaller than an ordinary Linux kernel**
- Example: yocto PROJECT

Size of Linux Kernel



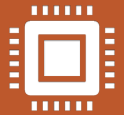
Embedded Linux File Systems

- File system must be as small as possible.
  - cramfs
    - A simple read-only file system that is designed to minimize size by maximizing the efficient use of underlying storage
    - Files are compressed in units that match the Linux page size
  - squashfs
    - A compressed, read-only file system that was designed for use on low memory or limited storage size environments
  - jffs2
    - A log-based file system that is designed for use on NOR and NAND flash devices with special attention to flash-oriented issues such as wear-leveling
  - ubifs
    - Provides better performance on larger flash devices and also supports write caching to provide additional performance improvements
  - yaffs2
    - Provides a fast and robust file system for large flash devices

Advantages of Embedded Linux

- Vendor independence
- Varied hardware support
  - Linux support for a wide range of processor architectures and peripheral devices
- Low cost for development and training
- The use of Linux provides all of the advantages of open source software

μClinux



μClinux (microcontroller Linux) is an open-source Linux kernel variation targeted at microcontrollers and other very small embedded systems

The design philosophy for μClinux is to slim down the operating environment by removing utility programs, tools, and other system services that are not needed in an embedded environment

Differences Between μClinux and Linux

- **Linux** is a multiuser OS based on Unix. **μClinux** is intended for embedded systems typically with no interactive user
- **μClinux** does not support memory management
- The **Linux** kernel maintains a separate virtual address space for each process. **μClinux** has a single shared address space for all processes
- **μClinux** only provides the vfork() system call for process creation

Designing a specific OS

Purpose-Built Embedded OS

- Fast and lightweight process or thread switch
- Scheduling policy is real time and dispatcher module is part of scheduler
- Small size
- Responds to external interrupts quickly
  - minimizes intervals during which interrupts are disabled
- Provides fixed or variable-sized partitions for memory management
- Provides special sequential files that can accumulate data at a fast rate

Examples of ad-hoc embedded OS

- eCos
- TinyOS

Timing Constraints

The kernel

- Provides bounded execution time for primitives
- Maintains a real-time clock
- Provides for special alarms and timeouts
- Supports real-time queuing disciplines
- Provides primitives to delay processing by a fixed amount of time and to suspend/resume execution

TinyOS

https://github.com/tinyos/tinyos-main

- Streamlines to a very minimal OS for embedded systems
  - Core OS requires 400 bytes of code and data memory combined
  - Has become a popular approach to implementing wireless sensor network software
- Not a real-time OS
- There is no kernel
- There are no processes
- OS doesn't have a memory allocation system
- Interrupt and exception handling is dependent on the peripheral
- It is completely nonblocking, so there are few explicit synchronization primitives

TinyOS Components

- Embedded software systems built with TinyOS consist of a set of modules (called components), each of which performs a simple task and which interface with each other and with hardware in limited and well-defined ways
- The only other software module is the scheduler
- Because there is no kernel there is no actual OS

Examples of standardized components include:

- Single-hop networking
- Ad-hoc routing
- Power management
- Timers
- Nonvolatile storage control

Components and Tasks

- A software component implements one or more tasks
- Each **task** in a component is similar to a thread in an ordinary OS
- Within a component tasks are atomic
  - Once a task has started it runs to completion

A task cannot	A task can
• Be preempted by another task in the same component and there is no time slicing • Block or spin wait	• Perform computations • Call lower-level components (commands) • Signal higher-level events • Schedule other tasks

TinyOS Scheduler

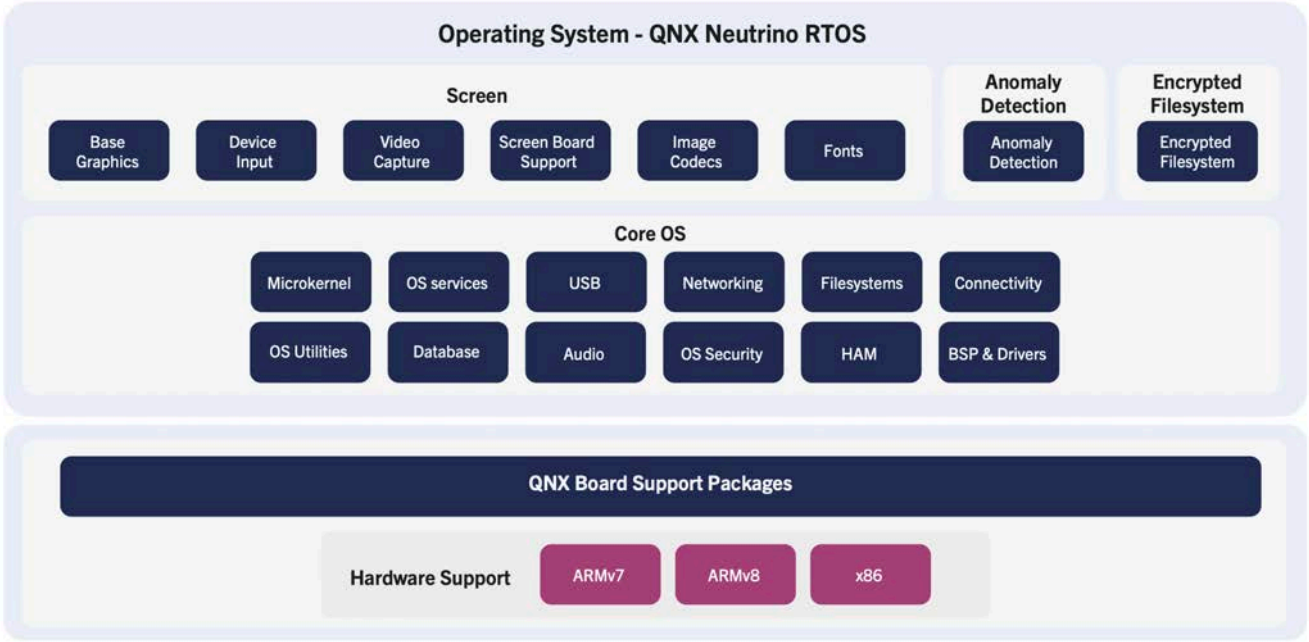
- Operates across all components
- Only one task executes at a time
- The scheduler is a separate component that must be present in any system
- Default scheduler is a simple FIFO queue
- Scheduler is power aware
  - Puts processor to sleep when there is no task in the queue

Blackberry QNX

- QNX
  - **QNX Neutrino RTOS**
    Realtime embedded system, microkernel design, and modular architecture that supports hundreds of POSIX commands, utilities, and programming interfaces
  - QNX OS for Automotive Safety
  - QNX OS for Medical

  - Wide adoption in the automotive sector

QNX Neutrino System Architecture



QNX Neutrino microkernel

- **thread services (**POSIX thread-creation)
- **signal services** (POSIX signal)
- **message-passing services**
  - the microkernel manages the exchange of messages between threads **synchronization services** (POSIX thread-synchronization)
- **scheduling services** (POSIX realtime scheduling policies)
- **timer services** (POSIX timer services)
- **process management services**

VxWorks and Integrity

- Wind River
  - **VxWORKS**
    RTOS with different profiles for different application scenarios.
    This OS equipped the MARS Curiosity mission (NASA)
- Green Hills
  - **Integrity**
    RTOS with a separation kernel for different markets

LYNX

- LYNX
  - **LYNXOS RTOS**
    LynxOS® is a deterministic, hard real-time operating system that provides POSIX-conformant APIs in a small-footprint embedded kernel. LynxOS provides symmetric multi-processing support to fully take advantage of multi-core/multi-threaded processors