

嵌入式操作系统概述

实时系统

- 一个实时系统是指计算的正确性不仅取决于程序的逻辑正确性，也取决于结果产生的时间，如果系统的时间约束条件得不到满足，将会发生系统出错。



术语

- 确定性（Determinism）：如果一个系统始终会为某个已知输入产生相同的输出，则该系统是确定性的
- 非确定性系统的输出具有随机变化特征
- 截止时限（Deadline）：截止时限就是必须完成某项任务的有限时间窗口，指明计算何时必须结束
-

http://design.ros2.org/articles/realtime_background.html

分类

- 硬实时 (Hard real-time) 软件系统有一组严格的截止时限，且错过一个截止时限就会认为系统失败
 - 示例包括：飞机传感器和自动驾驶系统、航天器和行星探测器
- 软实时 (Soft real-time) 系统会试图满足截止时限要求，但如果错过了某个截止时限也不会认为系统失败。但是，在这样一个事件中，软实时系统可能会降低其服务质量以改进其响应能力
 - 示例包括：用于娱乐的音频和视频传输软件（延迟是不可取的，但不是灾难性的）
- 准实时 (Firm real-time) 系统会将截止时限之后交付的信息/计算视为无效。与软实时系统一样，准实时系统在错过某个截止时限后不会认为系统失败，并且如果错过了某个截止时间，准实时系统可能会降低服务质量 (QoS)
 - 示例包括：财务预测系统、机器人装配线

嵌入式操作系统

- 伴随人工智能快速发展，嵌入式操作系统在智能系统发挥越来越大的作用
- 近年国内嵌入式操作系统市场规模整体保持稳定
 - 结合工信部最新数据显示，2022年我国嵌入式操作系统市场规模约469亿元

实时操作系统

- 实时操作系统（Real-Time Operating System, RTOS）是支持构建实时系统的操作系统
- 许多嵌入式系统都是实时系统，因此，用于该类的操作系统必须是实时操作系统
- 实时计算机系统同时需要实时运行的操作系统和提供确定性执行的用户代码
- 非实时操作系统上的确定性用户代码和实时操作系统上的非确定性用户代码都不会产生实时性能

RTOS and GPOS

- 相似的功能
 - 多任务级别
 - 软件和硬件资源管理
 - 为应用提供基本的OS服务
 - 从软件应用抽象硬件
-

RTOS从GPOS中分离出来的不同功能

- 嵌入式应用上下文中具有更好的可靠性
- 满足应用需要的剪裁能力
- 更快的特性
- 减少内存需求
- 为实时嵌入式系统提供可剪裁的调度策略
- 支持无盘化嵌入式系统，允许从ROM或RAM上引导和运行
- 对不同硬件平台具有更好的可移植性

RTOS关键要求

- 操作系统的时间行为必须是可预测的
 - 任何调度策略都必须是确定性的
 - 为了避免关键事件处理过程中的不可预测延迟，禁止中断的时间必须尽可能短
- 操作系统必须管理线程和进程的调度
- 一些系统要求操作系统管理时间
 - 对时间精度的要求可能会有所不同
 - 一些与环境的连接可用来获取精确的时间信息，如GPS或移动网络
- 操作系统必须是快速的
- 可靠性
- 简洁紧凑

为何使用RTOS

- 可被复用的标准软件组件
- 灵活性
- 响应时间

RTOS类别

- 快速专有内核
- 对于标准操作系统的实时扩展
 - RT_PREEMPT: Linux内核补丁，该内核补丁会将Linux的调度器修改为完全可抢占
 - Xenomai: 一个符合POSIX标准的协同内核（或管理程序），是一个可以提供与Linux内核协作的实时内核，Linux内核会被视为实时内核调度器的空闲任务（最低优先级任务）
 - RTAI: 一个协同内核（co-kernel）的替代解决方案

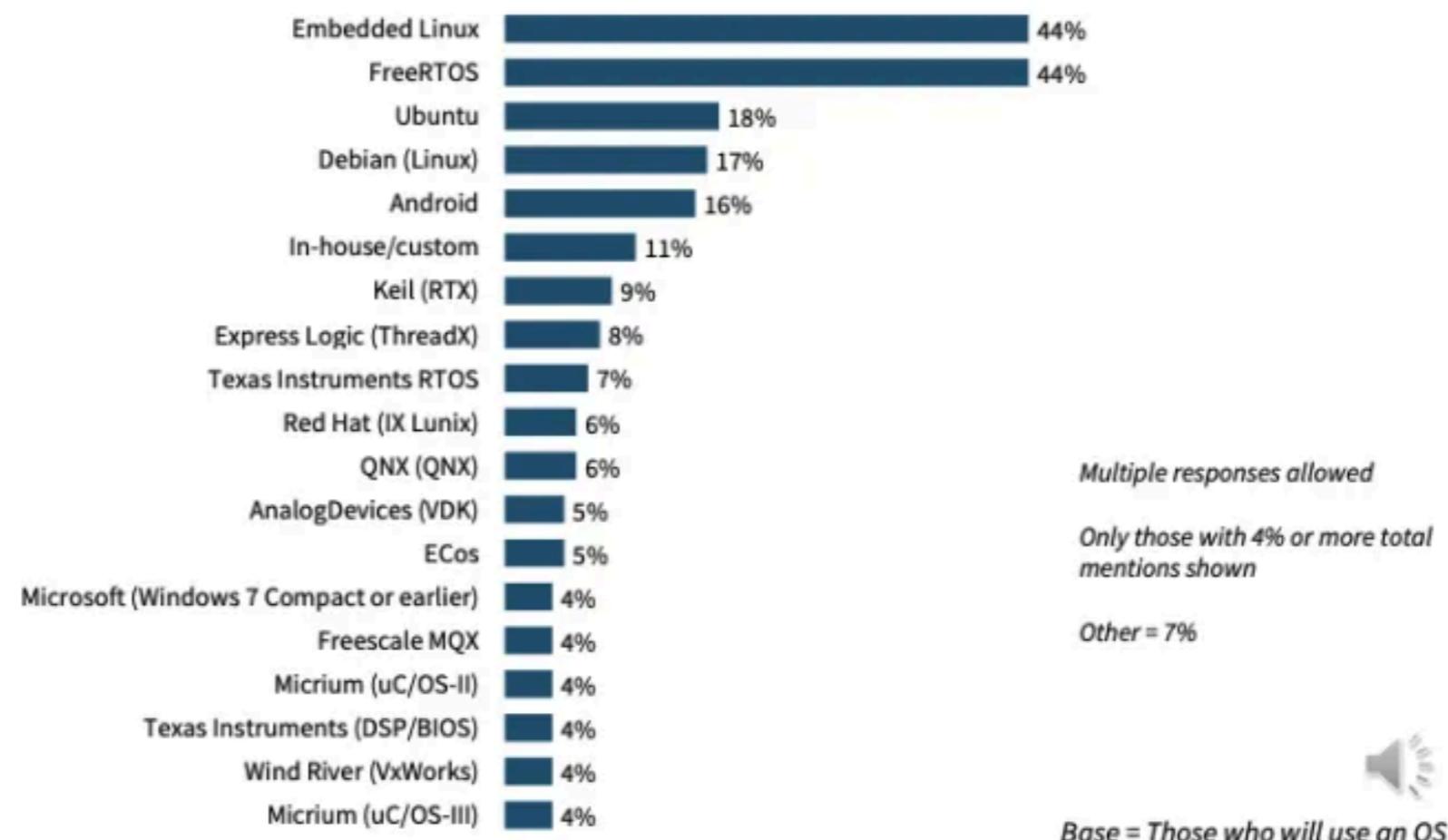
对于标准操作系统的实时扩展

- 优点：
 - 可配备标准操作系统的API，具有GUI、文件系统等
 - 标准OS的增强功能也可很快速地应用在嵌入式领域
 - 非实时进程不会影响实时进程
- 缺点：
 - 设备驱动程序存在一些问题，为避免冲突，有必要将设备划分为由实时进程处理的和由标准OS处理的两大类型
 - 实时进程不能使用标准OS的服务，因此，诸如文件系统访问、GUI等所有优秀特性对于实时进程一般是不可用的

最流行的嵌入式操作系统

Most popular embedded OSs – Embedded Linux, FreeRTOS and Ubuntu

Top 3 OSs are especially popular in APAC, while Embedded Linux is used more in the Americas



国产嵌入式操作系统

- 目前主流国产嵌入式操作系统包括
 - ▣ HUAWEI Lite OS鸿蒙操作系统
 - ▣ 可穿戴、智能家居、车联网
 - ▣ “道系统” 操作系统DeltaOS
 - ▣ 实时，可替换VxWorks6.8/6.9
 - ▣ 军用、工业互联网、物联网等
 - ▣ SylinxOS实时操作系统
 - ▣ 硬实时
 - ▣ 航空航天、国防安全、网络设备、汽车电子、机器人等
 - ▣ AliOS Things
 - ▣ 轻量级物联网嵌入式操作系统，云端一体
 - ▣ RT-Thread
 - ▣ 物联网操作系统，开源
 - ▣ 集RTOS内核、中间件组件和开发者社区于一体的技术平台，组件丰富

物联网操作系统

- 嵌入式操作系统历史悠久、门类繁多、用途广泛，不仅包括嵌入式实时多任务操作者系统(RTOS)、开源的Linux、机器人和路由器操作系统，还包括新型的物联网操作系统，以及边缘计算操作系统平台
- 物联网(IoT)源于大量廉价、小巧、节能的通信设备(或物)
- 从硬件上看，物联网是由异构硬件组成的

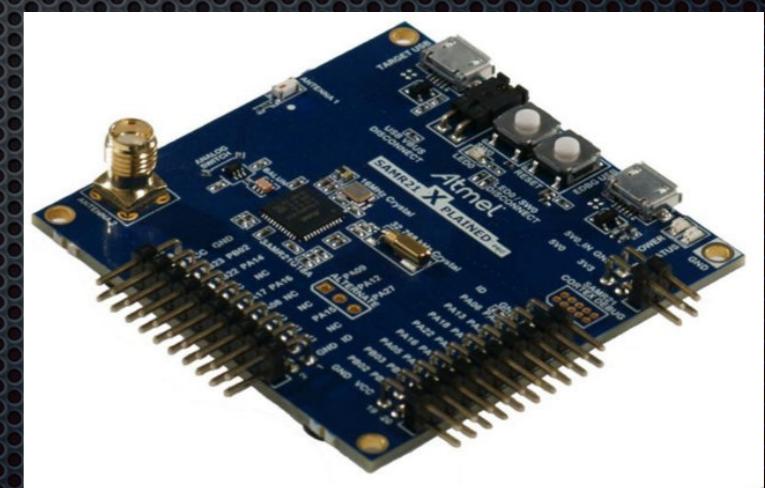
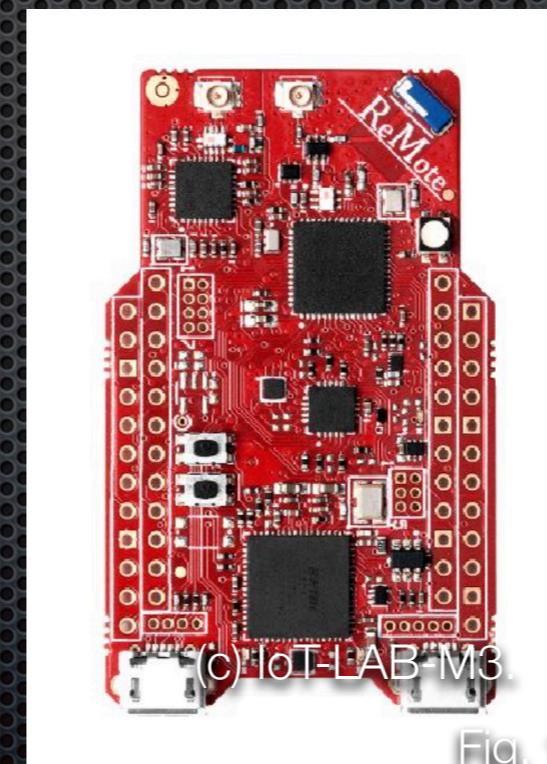
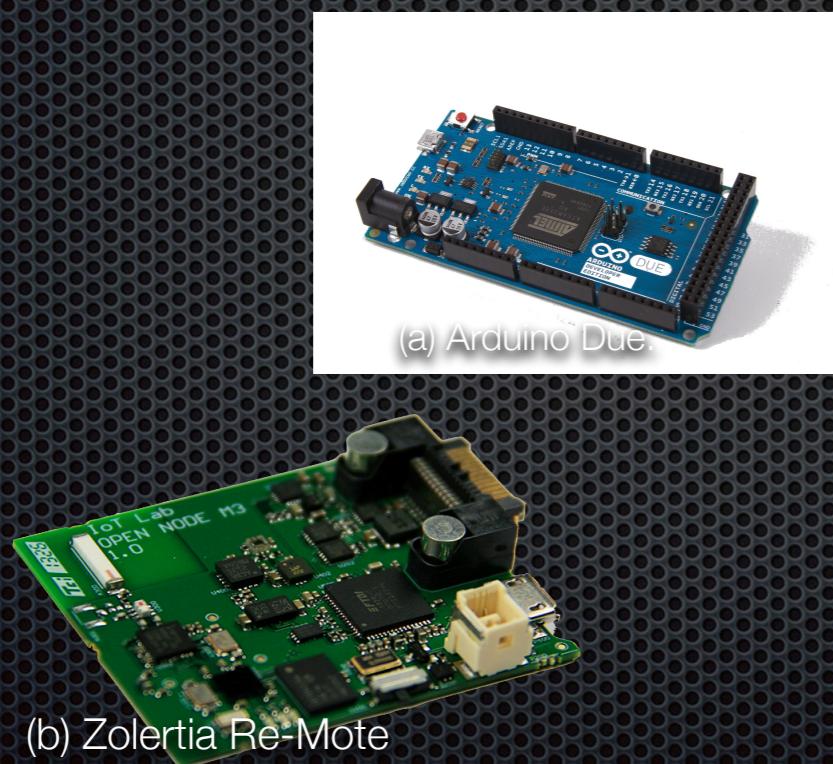


Fig. 1: Examples of low-end IoT devices.

物联网操作系统的要求-内存/异构硬件

- 内存占用小
 - 如下各方面需作出正确的权衡
 - 性能
 - 方便的API
 - 操作系统内存占用小
- 支持异构硬件
 - 物联网设备基于各种微控制器(MCU)架构和家族，包括8位(如Intel 8051/52, Atmel AVR)，16位(如TI MSP430)，32位(ARM7, ARM Cortex-M, MIPS32，甚至x86)架构，64位架构未来也可能出现。
 - 物联网设备可以配备各种各样的通信技术
 - 调查发现，在嵌入式环境中更换处理器频次比以前想象得更为常见

物联网操作系统的要求-网络连接/节能

- 网络连接
 - 物联网中使用的通信技术不仅包括各种各样的低功耗无线电技术(如IEEE 802.15.4、蓝牙/BLE、lora等)，而且还包括各种有线技术(如PLC、以太网或几种总线系统)
- 节能
 - 许多物联网设备使用电池或其他有限的能源
 - 在全球层面上，由于预计将部署的物联网设备的数量(数百亿)巨大，必须考虑节能
 - 物联网操作系统的一个关键需求是
 - 为上层提供节省能源的方案
 - 尽量利用这些功能本身，例如使用无线电周期工作技术，或尽量减少需要执行的周期性任务的数目

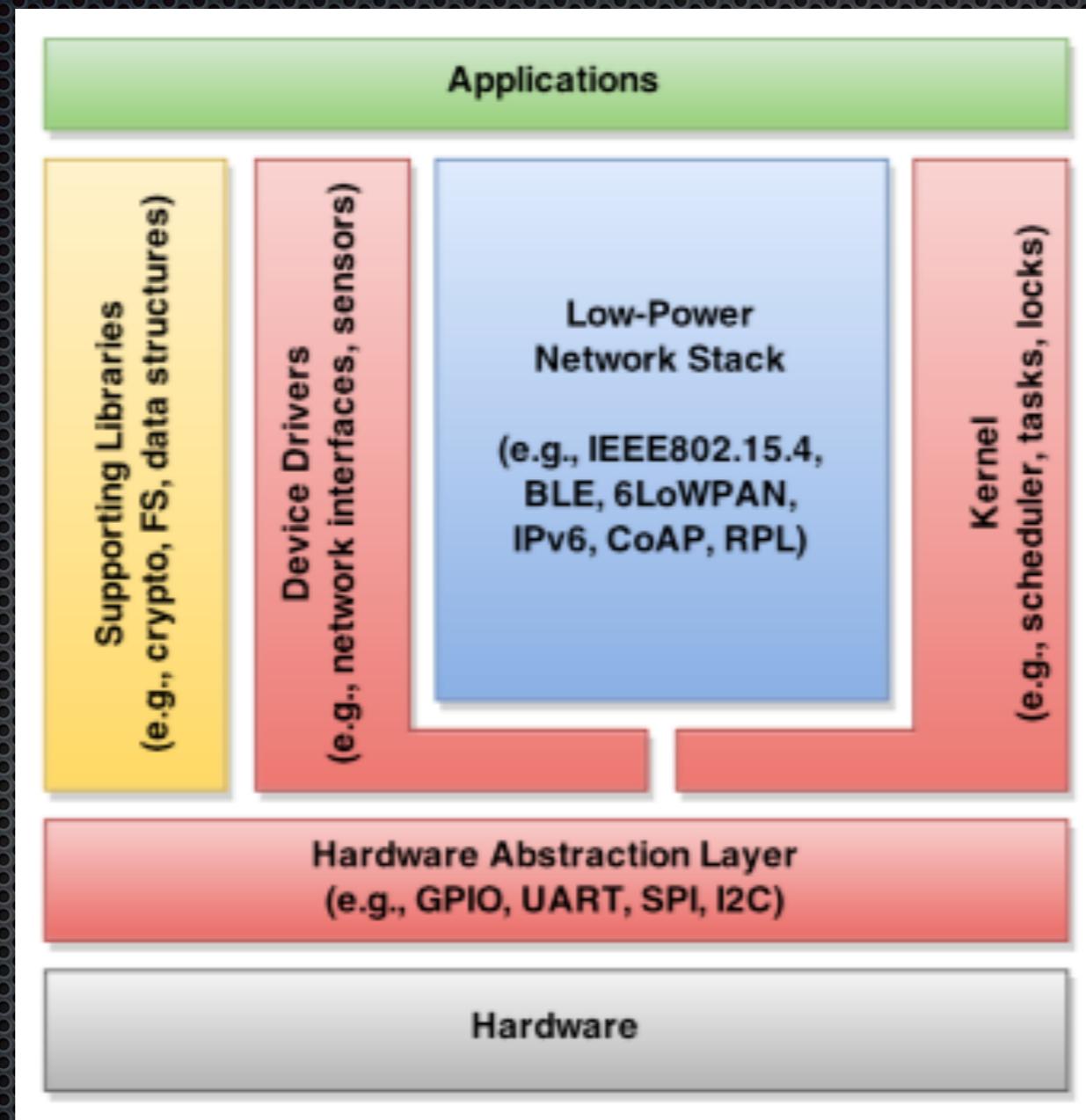
物联网操作系统的要求—实时

- 实时功能
 - 能够满足实时执行要求的操作系统称为实时操作系统，其设计目的是保证最坏情况下的执行时间和最坏情况下的中断延迟
 - 因此，对物联网通用操作系统的另一个要求是RTOS，这通常意味着内核函数必须使用一个确定的运行时进行操作
 - 日本实时操作系统的开放标准ITRON（Industrial the Real-Time Operation System Nucleus，工业实时操作系统中心）在这个领域很受欢迎，尽管它的目标主要是消费电子产品

物联网操作系统的要求-安全

- 安全
 - 一方面，一些物联网系统是具有生命安全影响的关键基础设施或工业系统的一部分
 - 另一方面，由于物联网设备与互联网相连，因此人们普遍期望它们能满足高安全性和隐私标准
- 除了主要的信任管理挑战之外，物联网安全挑战还包括物联网架构中各个部分的数据完整性、身份验证和访问控制
- 物联网操作系统的一个需求(和挑战)是提供必要的机制(密码库和安全协议)，同时保持灵活性和可用性

通用架构和模块化



低端物联网设备操作系统的典型组件，包括通用的低功耗IPv6协议栈

调度模型

- 两种调度器
 - 抢占式调度器
 - 非抢占(或合作)调度器

内存分配

- 内存是以静态还是动态的方式分配是一个重要的问题，这个选择也会影
响系统设计的其他标准
- malloc()和相关函数等函数在标准C库中通常以一种时间不确定性的方
式实现，因此将破坏任何实时保证。因此，为了给具有实时需求的应
用程序使用动态内存分配，操作系统必须提供特殊的确定性malloc()
的实现，如TLSF
- 动态内存分配需要在运行时处理内存不足等情况，这可能很难处理
- 基于堆的malloc实现通常会导致内存碎片，从而导致系统更快地耗尽
内存

网络缓存管理

- 有两种可能的解决方案
 - 复制内存(memcpy())
 - 从资源的角度来看是昂贵的
 - 在几个层之间传递指针
 - 产生了谁负责分配内存的问题

编程模型

- 物联网操作系统领域的典型编程模型可分为
 - 事件驱动系统
 - 多线程系统

编程语言

- 对于一个操作系统的编程语言，主要的选择有
 - 一种标准的编程语言，通常是ANSI C或C++
 - 特定于操作系统的语言

驱动模型和硬件抽象层

- 灵活、合理方便的驱动接口对于物联网操作系统至关重要
 - 物联网系统将以多种方式与环境交互
 - 被动的方式，通过各种传感器进行感知
 - 通过执行器的主动方式，如电机或照明系统
 - 因此，这些系统的mcu通常配备各种不同的外围设备，如ADC/DAC、接口(如SPI、I2C、CAN总线或串行线)和GPIO

调试工具

- 编程语言的选择也预先决定了可能使用的工具，包括用于调试的工具，完善的工具链通常包括相应的调试工具
 - 例如，围绕着GNU编译器集合(GCC)的工具链，包含GNU调试器(GDB)
- 为了运行实时调试系统，目标板必须提供适当的接口，如JTAG或Spy-Bi-Wire
- 一个常见的辅助工具是使用printf()和类似的工具在串行接口上进行简单的调试，例如USART
- 在某些情况下，甚至一个简单的LED闪烁算法有时也可以作为原始的调试替代

特征集

- 操作系统可以分为内核和更高级别的功能
 - 通常内核提供一个调度器、一个任务模型、互斥(互斥)和其他形式的同步以及计时器，如果操作系统支持多线程，API通常也包含IPC的函数
 - 在较高层可以找到系统库，如shell、日志、密码函数或网络栈，由于IoT设备上通常缺少mmu，这类应用程序和应用程序库通常会与内核操作运行在相同的地址空间中，因此会降低系统的稳定性
- 除了网络协议之外，在低端物联网设备的操作系统中，更高层的特性还包括空中更新OTA、动态加载和链接，或用于轻量级加密和解密的库

测试

- 物联网系统测试的具体挑战来自于
 - 这些系统的分布式特性
 - 典型的嵌入式以及常常是受限系统的事 实
- 处理与硬件相关的测试部分(如设备驱动程序的测试)的一种广泛使用的方法是使用硬件仿真工具，如MSPSim或Emul8

非技术属性

- (开放的)标准
- 认证
- 文档
- 代码成熟度
- 代码许可
- 操作系统提供商

(开放的) 标准

- 开发标准化api(如可移植操作系统接口POSIX, 由IEEE和Open Group指定), 以简化操作系统之间的软件移植
- 在低端物联网设备上, 由于软件大小的限制, 设计实现类似Linux等通用操作系统的标准API可能会很困难(事实上, 即使在pc上, 也很少有操作系统能声称完全符合POSIX)
- 标准不仅在系统层面很重要, 在网络层面也是不可避免的
 - IETF

证书

- 对于某些用例，特别是对于诸如建筑自动化等应用程序中的关键系统，系统的关键属性包括实时功能、健壮性或确定性
- 在这些情况下，通过独立机构进行认证成为操作系统不可避免的要求
 - IEC 61508标准，该标准名为“电气/电子/可编程电子安全相关系统的功能安全”
 - “IPv6 Ready” Logo测试认证，IPv6论坛发起的一个国际测试认证计划
 - 验证IPv6协议一致性和互连互通性，提供公正性测试规范
 - IPSO (IP for Smart Objects) 联盟合规和认证程序
 - 旨在推动IP协议作为网络互联技术用于连接传感器节点或者其它的智能物件以便于信息的传输

文档

- 完整且易于理解的文档对于任何软件都是非常重要的。对于操作系统来说，这个要求变得更加重要，因为操作系统是运行在系统上的每个软件的基础
- 对于嵌入式软件来说，对完整文档的需求更加迫切
 - 因为受到各种约束条件，这些软件通常不得不做出妥协，而这些妥协的原因乍一看很难把握
- 对于完全文档化的代码(但不一定是最有意义的度量)，一个典型的指标是每行代码的文档百分比

代码成熟度

- 比文档质量更难衡量的是软件的成熟度
 - 一个非常粗略的指标是项目的年龄与贡献者和用户的数量相结合
- 虽然在许多情况下，认证主要是一种法律保障，但系统的实际健壮性和正确性要难以评估得多

许可证

- 一般来说，有三种许可证类型：
 - 收费
 - 开放源码
 - 例如BSD、MIT或Apache许可，给予开发者和用户高度的自由，通常比copyleft许可更容易被业界接受
 - 版权许可证
 - 例如GPL和LGPL

OS提供者

- 操作系统的代码可能以不同的形式和不同的实体提供(取决于所选择的许可类型)
 - 它可能是由实际开发软件的供应商提供的，也可能由第三方提供的，后者也可能提供商业支持。
 - 对于开源解决方案，代码通常由开发人员社区本身通过版本控制系统的存储库(如Git、Subversion或Mercurial)提供
 - 社区通常通过在线论坛、开放Issue跟踪管理系统和邮件列表为这类项目提供最佳支持
 - 这种支持在实践中是至关重要的，因此强烈建议选择一个当前活跃社区的开源项目，而不是一个没有活跃社区的开源项目，或者一个以前活跃社区的开源项目
 - 有时，不仅为商业操作系统，也为免费的开源操作系统，提供专业的软件咨询

物联网操作系统候选

- 开源操作系统
 - 闭源操作系统
 - 物联网的其他软件库或中间件
-
- 如果没有另外提到，所有的操作系统都是用C编程语言编写的，而一些特定于硬件的部分可以用汇编语言实现

开源操作系统

name	architecture	scheduler	programming model	targeted device class ^a	supported MCU families or vendors	programming languages	license	network stacks
Contiki	monolithic	cooperative	event-driven, Protothreads	Class 0 + 1	AVR, MSP430, ARM7, ARM Cortex-M, PIC32, 6502	C ^b	BSD	uIP, RIME
RIOT	microkernel RTOS	preemptive, tickless	multi-threading	Class 1 + 2	AVR, MSP430, ARM7, ARM Cortex-M, x86	C, C++	GPLv2	gnrc, OpenWSN, ccn-lite
FreeRTOS	microkernel RTOS	preemptive, optional tickless	multi-threading	Class 1 + 2	AVR, MSP430, ARM, x86, 8052, Renesas ^c	C	modified GPL ^d	None
TinyOS	monolithic	cooperative	event-driven	Class 0	AVR, MSP430, px27ax	nesC	BSD	BLIP
OpenWSN	monolithic	cooperative ^e	event-driven	Class 0 – 2	MSP430, ARM Cortex-M	C	BSD	OpenWSN
nuttX	monolithic or microkernel	preemptive (priority-based or round robin)	multi-threading	Class 1 + 2	AVR, MSP430, ARM7, ARM9, ARM Cortex-M, MIPS32, x86, 8052, Renesas	C	BSD	native
eCos	monolithic RTOS	preemptive	multi-threading	Class 1 + 2	ARM, IA-32, Motorola, MIPS ...	C	eCos License ^f	lwIP, BSD
uClinux	monolithic	preemptive	multi-threading	>Class 2	Motorola, ARM7, ARM Cortex-M, Atari	C	GPLv2	Linux
ChibiOS/RT	microkernel	preemptive	multi-threading	Class 1 + 2	AVR, MSP430, ARM Cortex-M	C	Triple License ^g	None
CoOS	microkernel RTOS	preemptive	multi-threading	Class 2	ARM Cortex-M	C	BSD	None
nanorK	monolithic (resource kernel)	preemptive	multi-threading	Class 0	AVR, MSP430,	C	Dual License	None
Nut/OS	monolithic	cooperative	multi-threading	Class 0 + 1	AVR, ARM	C	BSD	native

TABLE I: Overview of potential open source OSs for the IoT

闭源操作系统

- ThreadX
- QNX
- VxWorks
- PikeOS
- embOS
- Nucleus RTOS
-

关注开源操作系统

- 下面将重点关注开源操作系统，原因如下
 - 通过在物联网设备上运行代码的透明度实现安全性和可信赖性
 - 在多方之间分摊开发成本(类似于Linux)的预期需要

与物联网相关的操作系统分类

- 事件驱动的操作系统
- 多线程操作系统
- 纯RTOS

事件驱动OS

- 这是最初针对无线传感器网络领域开发的操作系统最常见的方法，例如contiki或TinyOS
 - 该模型的关键思想是，系统上的所有处理都是由(外部)事件触发的，通常由中断发出信号
 - 内核大致相当于一个无限循环，处理同一上下文中发生的所有事件
 - 这样的事件处理程序通常会运行到结束
- 虽然这种方法在内存消耗和低复杂度方面是有效的，但它对程序员施加了一些重要的约束
 - 例如，并不是所有的程序都可以容易地表示为有限状态机
- 这类操作系统包括contiki、TinyOS和OpenWSN

多线程OS

- 多线程操作系统通常会引入一些内存开销，这是由于堆栈预留空间造成的，而运行时开销则是由于上下文切换造成的
- 属于这一类的操作系统包括RIOT、nuttX、eCos或ChibiOS
 - 由于更专注于物联网需求(据我们所知)，RIOT可以说是这类操作系统的一个很好的代表



纯RTOS

- 在工业/商业环境中，RTOS主要关注实现实时保证的目标，正式的验证、认证和标准化通常是非常重要的
- 为了允许模型检查和形式化验证，这些操作系统中使用的编程模型通常会对开发人员施加严格的约束
 - 这些限制常常使操作系统变得相当不灵活，并且移植到其他硬件平台可能变得相当困难
- 属于这一类别的物联网设备的操作系统包括FreeRTOS、eCos、RTEMS、ThreadX和一系列其他商业产品(通常是封闭源代码)
 - 据我们所知，FreeRTOS是最著名的物联网设备开源RTOS，因为它在各种环境中得到了更广泛的应用

各类别代表的关键特征

name	category	MCU w/o MMU	< 32 kB RAM	6LoWPAN	RTOS scheduler	HAL	energy-efficient MAC layers
Contiki	event-driven	✓	✓	✓	✗	✓	✓
RIOT	multi-threading	✓	✓	✓	✓	✓	✗ ^a
FreeRTOS	RTOS	✓	✓	✗ ^b	✓	✗	✗ ^c
uClinux	multi-threading	✓	✗	✓	✗	✓	✗
Android	multi-threading	✗	✗	✗	✗	✓	✗
Arduino	other	✓	✓	✗	✗	✓ ^d	✗

Ref.

- Oliver Hahm, Emmanuel Baccelli, Hauke Petersen, Nicolas Tsiftes. Operating Systems for Low-End Devices in the Internet of Things: a Survey. <https://hal.inria.fr/hal-01245551>
- 嵌入式系统设计：CPS与物联网应用



Thanks!!!