Rust kernel 调研报告

小组成员：陆万航，王浩宇，邱浩宸，雷婷，段逸凡

关键词：Rust 、嵌入式系统 、SOC、内核

1.项目背景

1.1Rust语言背景

计算机领域有一个顽疾：软件的演进速度大大低于硬件的演进，软件在语言级别上无法真正利用多核计算带来的性能提升。创建Rust语言的目的是为了根治这个顽疾。

Rust是针对多核体系提出的语言，并且吸收一些其他动态语言的重要特性，比如不需要管理内存，不会出现Null指针等。作为一门极其讲究精致的系统级别 语言，Rust运行性能高、能避免几乎所有的段错误且保证线程安全。它瞄准的目标是取代C/C++，这些出现在单核时代的比较贴近底层的高级语言已经渐渐 展现出了自己的不足，等待着后来者的超越。

Rust由web语言的领军人物Brendan Eich（js之父），Dave Herman以及Mozilla公司的Graydon Hoare 合力开发，最初是作为Graydon Hoare 的私人项目在2006年出现的，而 Mozilla 于 2009 年开始赞助这个项目。第一个有版本号的 Rust 编译器于 2012 年 1 月发布。Rust 1.0 是第一个稳定版本，于 2015 年 5 月 15 日发布。Rust的发展和变化很快，每一两个月就有一个新的版本。到2018年1月4日，Rust已经发布到了1.23版本。

虽然Rust是一门年轻的语言，在TIOBE编程语言排行榜上只是在50名左右，在热门程度和用户规模上难以与Java、C、C++等语言媲美，但她凭借着自身的优良特性，仍然得到了许多程序员的青睐。在将来，Rust或许也能够成为一门被广泛使用的伟大的语言。

1.2嵌入式系统背景

1.2.1什么是嵌入式

依照IEEE（Institute of Electrical and Electronics Engineers）对嵌入式的定义，嵌入式是指用于控制，监视或者辅助操作机器和设备的装置。嵌入式系统可以看做是一个删减版的计算机系统，一般是以一 个特定的目标为中心，以计算机系统为基础，进行硬件的剪裁和软件的定制。大多数的嵌入式设备都只能依靠单个程序，而进行对机械设备的控制，而另外一些嵌入 式设备则配有专门的嵌入式系统，可以完成更加复杂的工作，我们这里主要关注的是第二种。

嵌入式的结构与计算机有类似之处，由硬件层，中间层，系统软件层和应用软件层组成。在其硬件层面主要包括：嵌入式微处理器，存储器，各种设备接口，IO口等等。这些部件就构成了一个完整的嵌入式核心控制系统。

硬件层的核心是嵌入式微处理器，它与计算机所使用的cpu最大的不同在于，微处理器的功能更加集成，将pc机上由cpu和若干板卡完成的工作集于一身，而 且在完成特定的工作时，有着较高的效率与可靠性。嵌入式微处理器的类型很多，据估计已超过1000种，分为不同的体系，比较出名的有ARM，MIPS，PowerPC，X86和SH等。

嵌入式的存储器与计算机的区别较小，拥有cache，rom，ram和一些大容量的存储器，如硬盘，

NAND Flash,SD卡等。一些处理器将cache全部集成在微处理器内，以提高数据传输的速率。

因为嵌入式一般是用来控制一些机械结构或者一些特定功能的小配件，所以在嵌入式上一般有着比计算机更为丰富的通用设备接口，如A/D（模/数转换接口）、 D/A（数/模转换接口），I/O接口有RS-232接口（串行通信接口）、Ethernet（以太网接口）、USB（通用串行总线接口）、音频接口、VGA视频输出接口、I2C（现场总线）、SPI（串行外围设备接口）和IrDA（红外线接口）等。

在软件层与硬件层中间，是中间层，又称为硬件抽象层（HAL）或板级支持包（BSP）。

1.2.2什么是嵌入式操作系统

嵌入式操作系统，顾名思义，指用于嵌入式的操作系统，是一种拥有特定功能，用于大型机械，电气设备的计算机系统，通常拥有实时计算的特点。[2]

在早期的时候，嵌入式系统上的软件并不复杂，一台嵌入式设备只需要完成一个小任务即可。但随着技术的不断发展，嵌入式微处理器由8位到16位再到32位， 整个系统也变得越来越庞大和复杂。这就需要有一个操作系统对微处理器进行管理和提供应用编程接口（API）。于是，实时多任务内核（real-time kernel）在20世纪70年代末应运而生。进入20世纪80年代，嵌入式系统应用开始变得更加复杂，仅仅只有实时多任务内核的嵌入式操作系统已无法满 足以通信设备为代表的嵌入式开发需求。最初的实时多任务内核开始发展成一个包括网络、文件、开发和调试环境的完整的实时多任务操作系统（称为RTOS）。 到了20世纪90年代，嵌入式微处理器技术已经成熟，除了传统的x86处理器，以ARM7/9为代表的嵌入式处理器开始流行起来，这也让以Linux为代 表的通用操作系统进入了嵌入式系统应用这个领域，一些针对资源受限硬件的Linux发行版本开始出现，也就是我们所说的嵌入式Linux。进入2000年 以后，Android开始被广泛地应用在具有人机界面的嵌入式设备中。近来，物联网操作系统又以崭新的面貌进入了人们的视野。

嵌入式操作系统也是操作系统，所以操作系统所拥有的任务，其也必须具备其功能。如进程，通信，调度，内存管理等。近些年来人们对嵌入式设备的要求越来越高，嵌入式操作系统又逐渐多了一些功能，如多核、虚拟化和安全的机制，以及完善的开发环境和生态系统.[1]

1.2.3嵌入式系统的分类

嵌入式系统的分类有很多种，一般从硬件和软件两方面来对其进行划分。

从硬件上看，嵌入式系统可分为嵌入式微处理器，嵌入式微控制器，嵌入式DSP处理器，嵌入式片上系统等四类。

嵌入式微处理器（MPU）是从CPU演变而来，著名的ARM架构和MIPS架构就是这个种类，其特点是拥有较高的性能，与较高的加个。

嵌入式微控制器（MCU），典型代表正是大名鼎鼎的单片机。单片机的特点是拥有各种各样的外设插口，外设资源极其丰富，所以由此得名微控制器。

嵌入式DSP处理器（EDSP），DSP全称digital signal processor，专门用来信号处理，在数字滤波，FFT，频谱分析等方面拥有自己的优势。

嵌入式片上系统（SoC），近年来炙手可热的嵌入式系统，有着空前的集成性，在一个芯片上集成一个完整的系统。片上系统是替代集成电路的主要解决方案，是当前微电子芯片发展的必然趋势。

从软件上来看，主要分为实时操作系统和分时操作系统，其中实时操作系统又分为硬实时操作系统和软实时操作系统。由于嵌入式系统往往是为执行特定功能而设计 的，那么必须要保证严格按照时序执行。实时操作系统就好比汽车的安全气囊系统，如果没有在固定的时间内执行释放气囊这个任务，将酿成极大的悲剧，所以实时 操作系统的重要性越来越体现出来。

1.2.4嵌入式系统的发展历史

公认的最早的现代嵌入式系统应该可以追溯到1965年由MIT的Charles Stark Draper研制的Apollo Guidance Computer，其目的是在阿波罗计划中为飞船进行信号的传递和航天器控制。嵌入式系统的第一次大规模应用是用于民兵导弹的Autonetics D-17制导计算机。

嵌入式系统的出现是历史的必然。在上世纪六七十年代，计算机是大型的，在特殊机房内的使用的大型昂贵设备。而微型处理器以其小型，廉价，高可靠性的特点， 在一出现的时候就引起了控制专业人士的兴趣。随后嵌入式系统就被大规模的应用在了大型舰船，宇宙飞船，军事等方面。[3]

起初，计算机和嵌入式的发展是密不可分的，人们往往想把计算机高速处理海量数据与嵌入式的小巧的优点共同发展，甚至想将计算机嵌入大型设备，但是众多小型设备如家用电器等根本无法实现嵌入。

为解决这个难题，在20世纪末，计算机系统与嵌入式系统进行了专业化的分工发展。这一里程碑事件促进了双方的飞速发展。计算机领域迅速的从 286,386,486一直发展到奔腾，实现了历史性的飞跃。与此同时，嵌入式系统继续了其单芯片的特点，迅速将传统电子系统转向了智能化的现代电子系统。

2.立项依据

2.1 C语言编写的内核存在的问题(实例)

2.1.1 内核的安全性问题

在操作系统编程中，C语言凭借着自身跨平台、兼容性好、容易嵌入汇编代码、运行速度快、使用者众多等优点，成为了首选的语言。许多著名的操作系统内核，如Linux、Windows等，都是由C语言编写的。然而，诞生于上世纪70年代的C语言，其自身有着许多的问题。

安全性是C的一个致命缺陷：C不对数组越界、数值溢出等问题做任何检查，指针也可以随意使用，手动管理的内存容易产生泄漏等问题。尽管这些设计使得C的灵 活性大大提高，运行速度比起其他语言也有着不小的优势，但是，C程序的安全性得不到保证，即使是它的编写者也不能担保自己的程序没有漏洞。

显然，使用C编写的操作系统内核也不免有这些问题，而内核的漏洞比起应用程序的漏洞要严重许多。后者最多使进程崩溃，前者却可能造成系统崩溃、信息泄露、文件丢失等诸多问题。下面，就以Linux系统漏洞为例，揭示内核的安全性问题：

2003年9月份，Linux内核开发人员发现了do_brk()边界检查不充分漏洞，并在9月底发布的Linux kernel 2.6.0-test6中对其进行了修补。

该漏洞被发现于brk系统调用中。brk系统调用可以对用户进程的堆的大小进行操作，使堆扩展或者缩小。而brk内部就是直接使用do_brk()函数来 做具体的操作，do_brk()函数在调整进程堆的大小时既没有对参数len进行任何检查（不检查大小也不检查正负），也没有对addr+len是否超过 TASK_SIZE做检查。这样我们就可以向它提交任意大小的参数len，使用户进程的大小任意改变以至可以超过TASK_SIZE的限制，使系统认为内 核范围的内存空间也是可以被用户访问的，使得普通用户可以访问到内核的内存区域。通过一定的操作，攻击者就可以获得管理员权限。这个漏洞极其危险，利用这 个漏洞可以使攻击者直接对内核区域操作，可以绕过很多Linux系统下的安全保护模块。

NULL指针解引用也是最常见的漏洞之一。指针即包含内存中某一变量的地址值，当指针解引用时，即可获取内存地址中存放的变量值。一个静态未初始化的指 针，其内容为 NULL即（0x0）。在内核代码中NULL值常在变量初始化、设置为缺省值或者作为一个错误返回值时使用。在系统中，虚拟地址空间分为两个部分，分别称 为内核空间和用户进程空间，当内核尝试解引用一个NULL指针时，由于进程在执行指针解引用时，没有检查其合法性，若黑客在用户态将NULL地址设置为可 执行并注入恶意代码，则内核代码将会执行NULL地址的恶意指令。比如，利用CVE-2010-4258漏洞就可以在任意内存位置写0。

当然，Linux系统的漏洞远不止上面所说的这一些。尽管官方在不断地修补漏洞，但每年都有新的漏洞被发现并报告。实际上，Linux内核的源代码在发布 前都会经过重重审核，代码的编写者也都是经验丰富的程序员，但仍无法完全避免安全性问题，只能不断地修复已知漏洞，以提高系统安全性。

其实，安全性并不是Linux一家的问题，Windows的永恒之蓝漏洞曾导致了影响全球的wannacry勒索病毒大爆发事件，造成严重的危机管理问题，影响了金融、能源、医疗等诸多领域，造成损失逾80亿美元。可见，提高操作系统的安全性是一个重要的课题。

在嵌入式领域，由于物联网的发展，几乎所有的嵌入式设备都有了互联网接入能力，体系结构从封闭到开放，形成了互联网中不安全因素对嵌入式系统的入侵渠道， 物联网系统的安全性设计上升到空前高度。比如目前飞速发展的汽车电子领域，自动驾驶时代即将来临，但安全性的要求也随之提高，如果黑客通过网络对自动驾驶 系统进行攻击，控制了行驶系统，甚至可能人工制造车祸，令人不寒而栗。操作系统安全关系着我们的生命安全，这更加凸显出操作系统安全的重要性。

2.2 Rust语言的优良特性

Rust 是一门年轻的语言，从创立之初，就瞄准了C和C++，从这两门语言中借鉴了许多优点，并规避了这两门语言的许多问题。在底层的内核开发方面，古老的C语言越来越跟不上时代的发展，而Rust注重于安全性、高效性和并发性，成为了一个有力的挑战者。

2.2.1 安全性

安全性的定义本身就是一个充满争议的话题。对于安全性的理解有许多，其中的一个理解是，不出现未定义的行为。

C本身的安全性有着很大的问题，而Rust解决了这些问题。事实上，在安全模式下，（unsafe{}代码块之外的代码），Rust几乎没有未定义的行为。

在内存管理上，C依靠的是人工手动的内存操作，从内存空间的开辟到释放全都要依靠程序员，因而在大型项目时很难避免内存的泄漏。而Rust在编译时通过 RAII(Resource Acquisition Is Initialization)实现资源自动释放。每一个对象都仅和一个变量绑定（严格的编译保证了这一点，使得运行时不用付出任何开销），当变量离开作 用域时，它绑定的资源也会被回收。[4]这样不但能提供内存安全，还提供数据和资源安全。

在并发上，Rust在语言层面支持了绿色线程 (Greenthreads)——Task。Task作为并发执行的单元，是用户空间的“线程”，创建和调度成本较低，可以大量共存。Task之间通过消息传递通信，没有直接共享数据，这种设计增强了安全性。[5]

为了确保在并发编程中的数据安全，Rust采用的更重要的方法是，某个时间点同时只允许多个读操作或一个写操作访问共享数据。

为了实现这一点，Rust引入了所有权的概念。一个资源的所有权只能属于一个变量，变量的生命周

期等于资源的生命周期。所有权可以在变量之间转移，但在任意时刻，都只有一个变量能修改资源，或是若干个变量可以读取资源，确保了数据安全。

宏是一个“展开后的”句法形式的速记，在有些时候，它可以替代函数的作用，而且拥有比函数更高的效率（在要求效率的场合，比如内核代码，可以看到宏的广泛 应用）。一些语言（如C）中的宏使用简单的文本替换来实现，会导致许多的问题，而Rust的卫生宏可以解决这个问题。

2.2.2 高效性

Rust语言在运行时做的工作极少，把大量的工作留到编译时完成，这使得Rust语言具有高效性。

以内存回收为例，垃圾回收GC是解决内存安全的最普通方式。然而，JVM在程序运行时自动运行着垃圾回收程序，监控着每一个对象的状态，显然会导致程序运行速度的下降。因此，Rust放弃了这种成熟的方式，转而采用RAII机制。

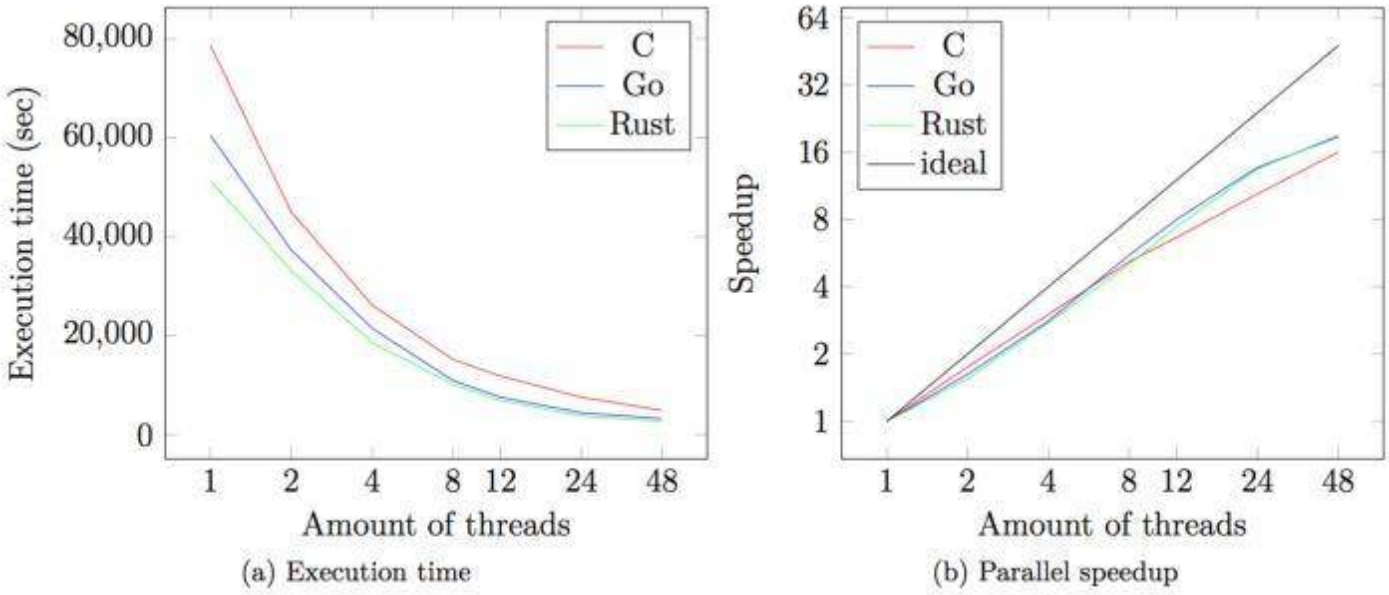
此外，Rust 还致力于实现“零开销抽象”，虽然有些抽象看起来更像一个高级语言的特性，但实际上并不会产生运行时开销。

特质（Trait）是Rust中的重要角色，它有些类似于接口。特质是一个告诉 Rust 编译器一个类型必须提供哪些功能的语言特性。而且特质之间还可以相互继承，实现一个子特质就必须先实现另一个父特质。

Rust还提供了枚举和模式匹配等许多功能，它能够做到许多if语句做不到的事情，大大方便了编程和开发。

上面介绍的一些抽象，更像是由一个类似于Java的高级语言所提供的，而不是Rust这样一个接近底层的语言。良好的抽象，会在使用的时候带来不少的方便，使得编程更加容易，这也是另一种意义上的高效。

实际的工作更加印证了上面的理论：来自德国汉堡大学的研究者做了一个关于不同语言在高性能计算中开发效率和性能的评估[6]，使用了C、Go和Rust语言做同一个项目，评估它们的效率和性能。下面是关于性能和开发效率方面的结果：



(a) Execution time (b) Parallel speedup

Figure 5.1.: Performance metrics across the various milestones

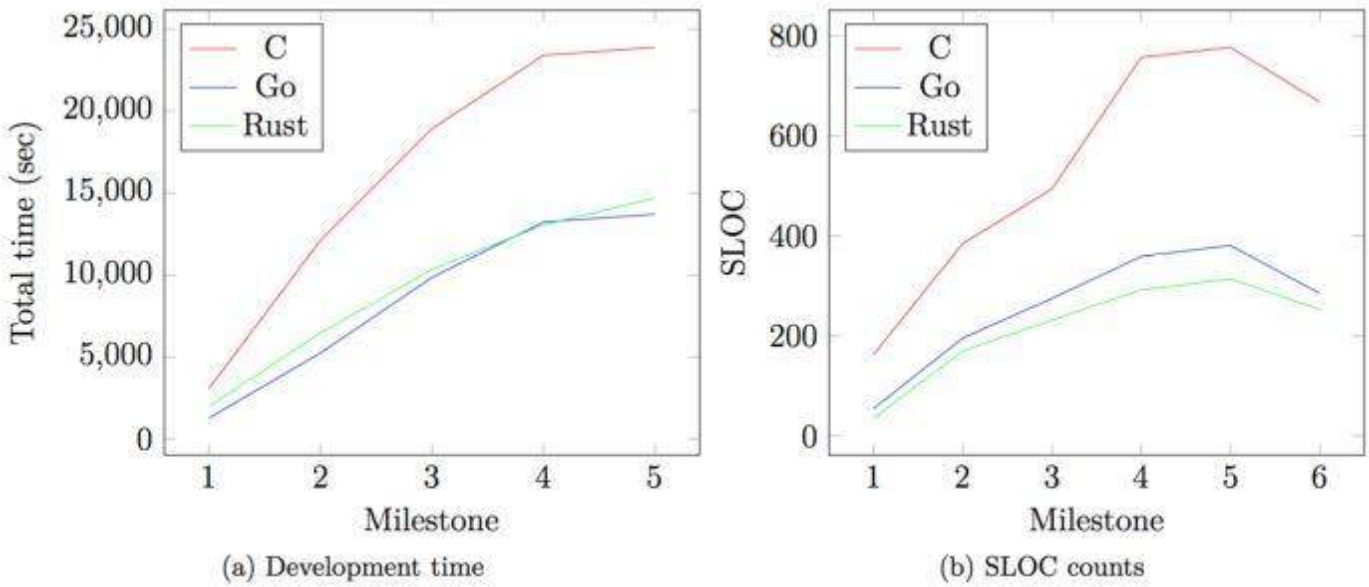


Figure 5.2.: Productivity metrics across the various milestones

综合上面的结果，可以发现，Go 和 Rust 都是高性能计算的有力竞争者，不过 Rust 的表现更好。这项工作清楚地揭示了Rust语言的高效性。

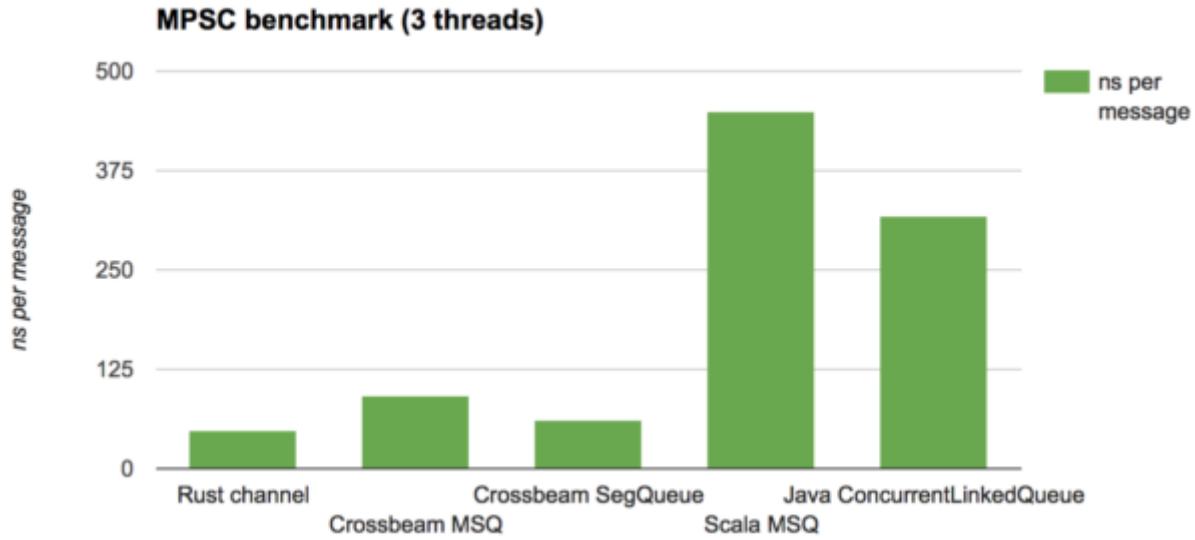
2.2.3 并发性

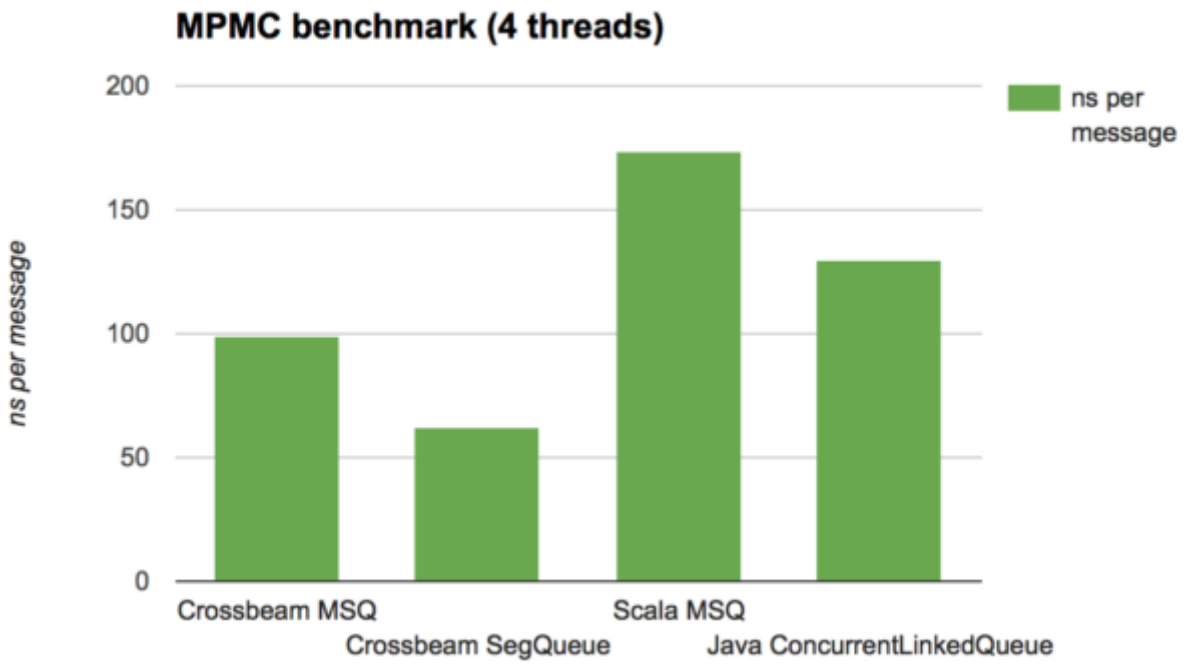
Rust 的内存安全功能也适用于并发环境，并发的 Rust 程序也会是内存安全的，并且没有数据竞争。Rust 的类型系统也能胜任对并发情形的处理，并且在编译时能提供强大而有效的方式去推论并发代码。

Rust禁止了不安全的并发代码，不是通过一个线程安全的方式维护的类型将不被允许在线程间传递。同时，Rust提供了标准的原子引用计数类型，可以在多线程间安全地访问。借此，编译器确保了内部计数的改变都是不可分割的操作，这样就不会产生数据竞争。

使用Rust能够建立一种对并发数据结构进行内存管理的API，它非常容易地实现类似GC那样无锁的数据结构，且比并发GC有更小更可预期的开销。[7]

Rust非常易于实现一个类似Java 的无锁队列，作者称之为Crossbeam, 并与Java进行了性能对比，下图中的Crossbeam MSQ是使用Rust Crossbeam编写的无锁队列：





上面两张图一个是MPSC，一个是MPMC：MPSC是多个生产者 一个消费者场景，两个线程重复发送消息，一个消息接收消息。MPMC代表多个生产者多个消费者场景，两个线程发送并且两个线程接收。

从上面对比看出，Rust Crossbeam性能无疑是胜出Scala或Java无锁队列。

2.3.嵌入式系统对安全性和实时性的要求

2.3.1 嵌入式系统对实时性的要求

嵌入式系统也是一个激励 - 运行 – 响应的电子系统。但是，它与嵌入对象交互和嵌入对象的事件过程相关，在与嵌入对象体系交互时，要满足事件交互过程的响应要求，一方面，嵌入式应用系统有十分可观的激励 - 响应时间ta，导致系统实时能力的降低；另一方面，由于嵌入对象体系的多样性，复杂性，不同的对象体系会提出不同的响应时间 t a 要求。因此，在嵌入式应用系统的具体设计中，必须考虑系统中每一个任务运行时，能否满足 ts<=ta的要求，这就是嵌入式系统的实时性问题[8]。

嵌入式系统由于是嵌入到对象体系中的一个电子系统，与对象系统密切相关，而形形色色的对象系统会有不同的响应时间ta要求，例如，超市的称重、计量、收银 机只要求有尽快的响应时间即可，即便有几秒的延迟，也不会影响用户的使用；但是，汽车内置的安全气囊却要求在发生碰撞时立即响应，且响应时间极短，否则会 导致相当严重的后果。在同样的动态信号采集系统中系统的响应时间与信号的动态特征有关，这些不同的嵌入式应用系统的不同响应要求，表现了嵌入对象响应要求 （ta）的多样性。

嵌入式应用系统的激励 - 运行 - 响应特性，形成了以软件运行时间ts 为主要内容的系统响应能力 T 。而软件运行时间ts 与指令速度、编程技巧、程序优化等有关，是一个应用系统实时能力的可变更性。因此 ta 的多样性要求与响应时间 ts 的可调整性,是嵌入式系统的实时性分析的基本出发点。根据嵌入对象 ta 的不同要求，调整 变更 ts大小 ，以实现 ta 的最佳化，是 嵌入式系统实时性设计的一项重要内容。

嵌入式系统的实时性不是一个快速性概念，而是一个等式概念，即能否满足 t s<=t a 的要求，而非

越快越好。因而，快速系统不一定能满足系统的实时性要求，而某些情况满足实时要求时，系统运行速度并不高。例如，满足温度采集实时性要求的嵌 入式系统，运行速度并不高；而许多高速运行的系统，未必能满足冲击振动的信号采集的实时性要求，快速性只是反映了系统的实时性能力而已。

快速性是系统实时能力的表现，当系统不能满足实时性要求时，必须提高系统的运行速度，然而，运行速度的提高必然带来系统的一些负面效应，如导致系统功耗加 大、电磁兼容性下降。因此，在设计一个具体的嵌入式系统时，在保证能满足实时性要求的条件下，应使系统的运行速度降到最低，以满足系统在功耗、可靠性、电 磁兼容性方面获得最佳的综合品质。

在一个嵌入式应用系统中，往往有许多过程环节。例如，一个典型的智能仪表就有信号采集、数据处理、结果显示、键盘输入等过程。这些过程往往是在不同的时间 与空间上进行，而且不同过程的实时性要求是不同的。键盘输入、结果显示是与人交互的，要满足人机交互的实时性要求；信号采集与对象系统信号的动态性密切相 关，必须满足由动态信号采集的实时性要求；而数据处理则会形成从动态信号采集到结果显示的时间延迟，影响到结果显示的实时性要求。因此一个优秀的实时系统 设计，必须研究系统中的每一个过程环节，满足每一个过程环节和整个系统的最佳实时要求。

对很多嵌入式系统来说，一个设计良好的实时操作系统(RTOS)可以让开发工程师把握系统执行任务或响应关键事件的时间，以满足系统实时性的要求。 一个好的RTOS支持开发人员能控制系统执行任何任务或对任何重要事件做出反应的时间，并且能够以一种可以预测并且完全一致的形式满足任务执行的时间要 求。

2.3.2 嵌入式系统对可靠性的要求

社会发展日新月异，物联网离百姓生活越来越近，目前很多运行在局域网甚至Internet上的产品如雨后春笋般涌向市场，如智能家居、安卓手机等。这些产 品在方便用户的同时也出现一些安全问题，系统置于网络上相当于暴露给所有人，故对嵌入式产品安全性研究刻不容缓。嵌入式产品由于尺寸、成本的约束注定不可 能从硬件部分提供更多的安全措施，故提升系统安全性应重点考虑内核方面的安全。下面我们将以linux内核安全性为例阐述当下嵌入式内核安全性方面存在的 问题。

开源的Linux内核无论在功能上或性能上都有很多优点，但Linux内核属于开源项目，缺少开发商的安全保证，所以需要使用者最大限度地提升系统的安全 性。根据实际情况裁剪Linux内核，并采取适当的安全措施可提升系统的安全性。掌握Linux核心技术、配合先进的安全模型、增强其安全性进而研发安全 的操作系统非常有必要。Linux的安全性问题从机制角度可以有以下几点：

1. Linux系统访问控制。Linux系统主流发行版本的访问控制属于自主访问控制；自主访问控制控制模式任何一个活动主体对应用户标识和组标识。显然自主访问控制模式的问题是忽略了用户的角色、程序可信性等安全信息，故不能提供有效的安全性保障。
2. Linux系统管理。Linux系统中用户可以分为两类，一类是普通用户，另一类是管理员用户。Linux系统管理员用户拥有系统所有权限，包括 用户管理、设备管理、审计管理和安全管理等；这样方便了管理员管理，易于用户操作，但是违背了“最小特权”管理原则。Linux系统的安全性只是建立在管 理员必须正确设置系统功能、且不被冒充和不存在安全漏洞等一系列假设的基础上。显而易见这样的系统存在巨大的风险，假如系统管理员被非法控制，系统将没有 安全性可言。
3. Linux系统日志。Linux系统中的日志功能的设计不是以系统安全为目标而是以内核调试为目的；此与系统安全审计有很大差距：第一，缺乏资源访问方面的记录；第二，不能详细记录系统发生的事件；第三，缺少必要数据分析与警告。

由此可见，嵌入式Linux内核自身的安全功能相当薄弱，对于安全性要求比较高的产品，则需要提高Linux内核的安全性[9]。

一般来说，在采用非实时操作系统(non-RTOS)的任何场合，也都可采用RTOS。但是，要找到一款具有完全相同应用编程接口(API)的匹配 RTOS相当困难。因此，许多传统的操作系统在其内嵌入了一个RTOS。例如，Lynux-Works LynxOS和Bluecat Linux共享一个Linux API。LynxOS是一款硬RTOS，而Bluecat是Linux的一个衍生产品。 大多数实时系统要求有较高的可靠性。

在一些重要的实时应用中，任何不可靠因素和计算机的一个微小故障，或某些特定强实时任务（即关键任务）超过时限，都可能引起难以预测的严重后果。为此，系 统需要采用静态分析和保留资源的方法及冗余配置，使系统在最坏情况下都能正常工作并避免损失。可靠性已成为衡量实时系统性能不可或缺的重要指标 。但是，假如RTOS崩溃，这些最终期限就不能被满足。因此，RTOS必须有高度可靠的防意外崩溃机制，即RTOS必须拥有在不需要重启的情况下，就可以 从故障中快速恢复的机制。

3.前瞻性/重要性分析

3.1.嵌入式系统的应用领域

3.1.1.嵌入式系统应用概述。

如上所述，嵌入式系统有小型、廉价、高可靠性等优点，因此，嵌入式系统有着相当广泛的应用。它的应用几乎涵盖了所有非PC的计算机系统，包括网络通信类产 品、工业、医疗、军工航天等诸多领域。此外，移动智能终端（如智能手机、平板电脑等设备），多采用嵌入式微处理器，有着很多嵌入式系统的特征，移动互联网 终端在广义上也依旧可归于嵌入式系统。[11]

总的来说，为了支撑这些产品的设计需求，作为核心的移动终端嵌入式微处理器需要具备高集成，高性能，低能耗的特征。为了支撑如智能手机一类的小型终端，移 动终端处理器必须将尽可能多的功能模块集中到一块芯片中。如何将主CPU、集成GPU、VPU等诸多部件集成到一块芯片上，同时保证高集成度和高可靠性正 是现代SoC（System on Chip，系统级芯片）设计的一个巨大挑战。[10]

3.2.2 嵌入式系统中的SoC架构。

传统的PC架构是基于北桥和南桥两个桥接芯片的，整个系统是集成在一块主板上的。而嵌入式SoC芯片往往将整个系统集成在一块硅片上。相比于PC的系统，SoC可以说是一个微型系统，可认为是计算机系统的一个子集。嵌入式系统的硬件架构抽象图如下所示：

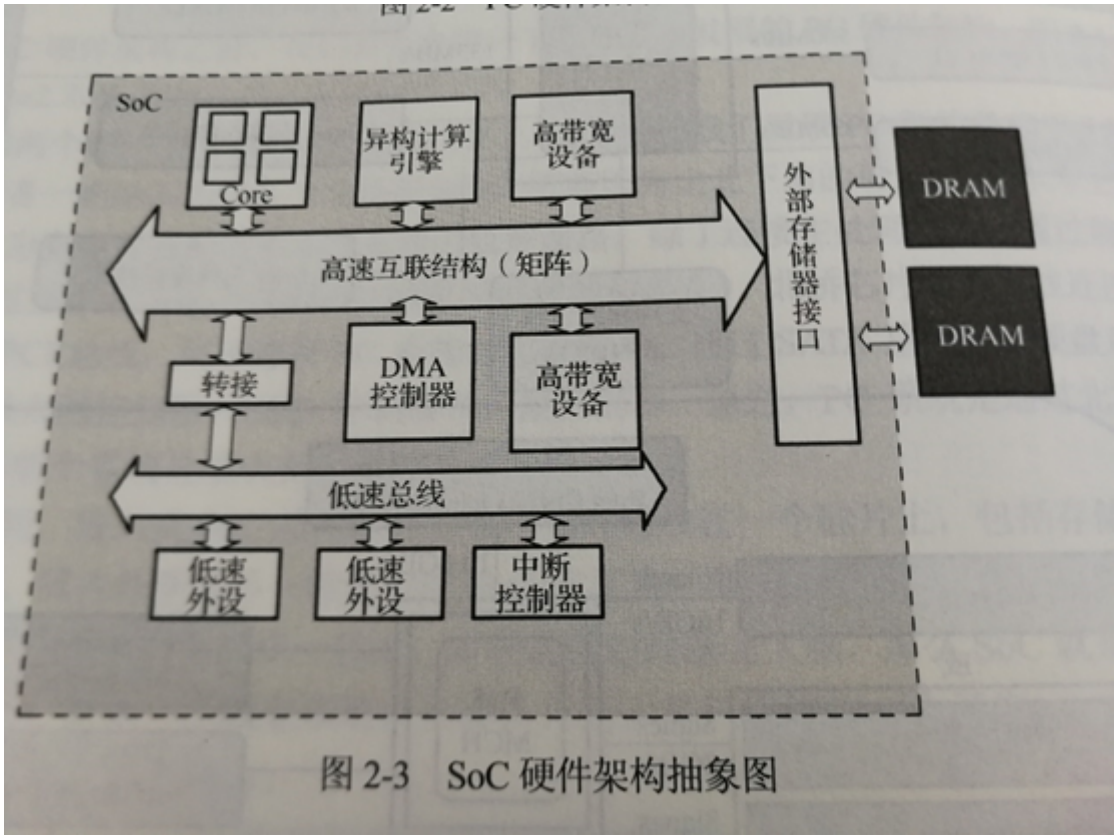


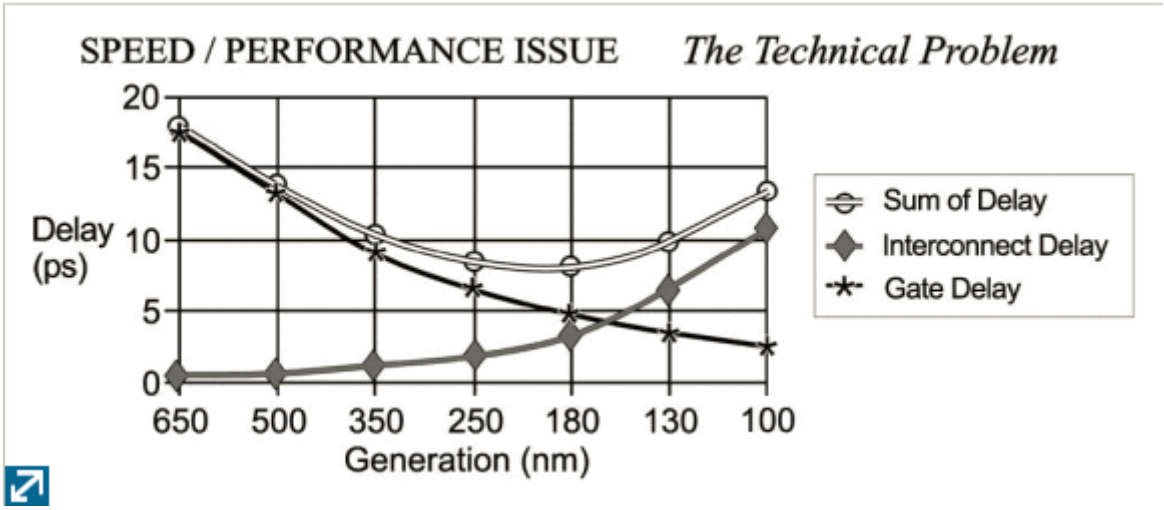
图 2-3　SoC 硬件架构抽象图

与普通的PC架构一样，SoC的核心也是CPU内核，随着技术的发展，其功能复杂性也是与日俱增。如上图所示，SoC的高速互联架构中，除了嵌入式CPU之外，还挂构了诸如DMA控制器、异构计算引擎等对带宽要求较高的高速处理设备。

嵌入式系统中，存储系统是另外一个非常重要的组成部分。在片上系统设计过程中，存储子系统制约了整个嵌入式系统的功耗、性能以及芯片内容，所以存储子系统 成为了嵌入式系统设计的核心成分。为了获得较高的访问性能，SoC常采用层次性的存储结构，即将最为频繁访问的部位置于最为顶层的存储器中，使之十分接近 处理器，从而加快处理速度。[10]

3.2.3 SoC未来的发展方向

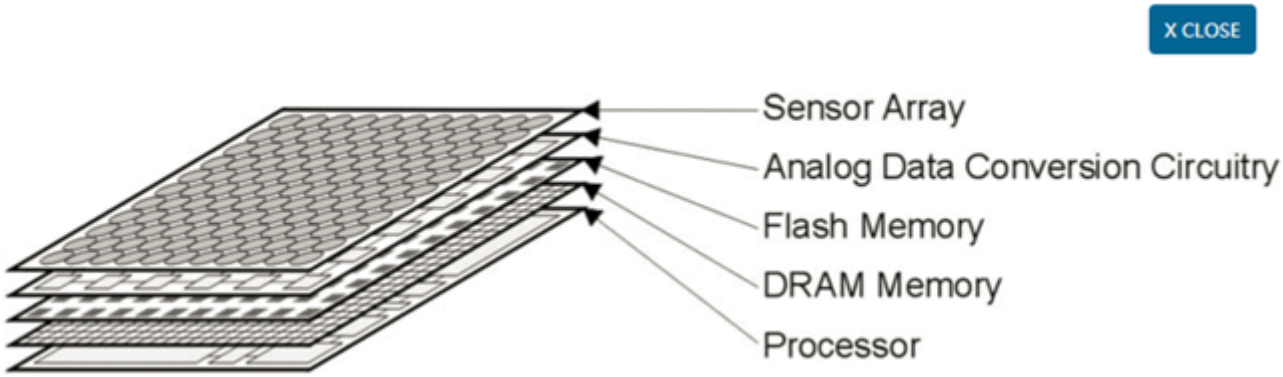
随着集成电路（Integrated Circuit）的不断发展，门电路的尺度也在不断减小。但是，受到物理规律的限制，逻辑门的传输延迟时间的降低速率也在不断降低（见下图）。而且，虽然 将集成电路的规模进一步缩小在物理意义上是可行的，但是随着集成电路规模的减小，芯片的经济成本也会随之上升，故而从经济角度看，这么做也是不合理的。所 以，传统的二维SoC的发展势必会因此受到限制，于是，有人提出了在三维模型中构造集成电路来解决这个问题。[11]



三维集成电路（3-D IC）并非将集成电路搭建在平面上，而是将其构造成球形，这样子便可以获得相当于传统二维集成电路十倍的电路量，其架构主要分为芯片堆块（chip tacking）、晶体管堆块（transistor stacking）、晶片模具堆块（die-on-wafer stacking）、晶片层堆块（wafer-level stacking）等四个主要步骤。

三维集成电路为嵌入式SoC提供了一个很好的发展方向。SoC中大部分空间被各种形式的嵌入式内存占据，由于芯片的空间被大量内存占据，故门电路必须被建 造成很小的尺寸（65nm左右），这是被芯片的空间所限制，而非门电路的数量和速度要求导致。所以可以将内存单独独立出来成为一层，从而使集成电路有着更 多的空间。此时，集成电路的尺寸可以扩展到110nm乃至180nm，从而显著地降低了成本。

而独立内存层的另一个好处就是，内存不必再和高频使用的晶体管共享进程了，所以可以构建一个真正的内存进程。这个内存甚至可以是DRAM而非SRAM。下图既是独立内存层的一个设计样例：



尽管3D建造工艺在当今的芯片生产中并不占主流地位，而且3D SoC是否能投入实际生产流水线中还要看其具体的投入产出比。但是，随着纳米技术的尺度降低带来的成本增加，3D SoC技术投入使用应该只是时间问题了。

3.2 Rust语言的发展前景

根据Rust官网发布的2017Rust调查报告[12]显示，Rust平台呈现出繁荣的趋势：

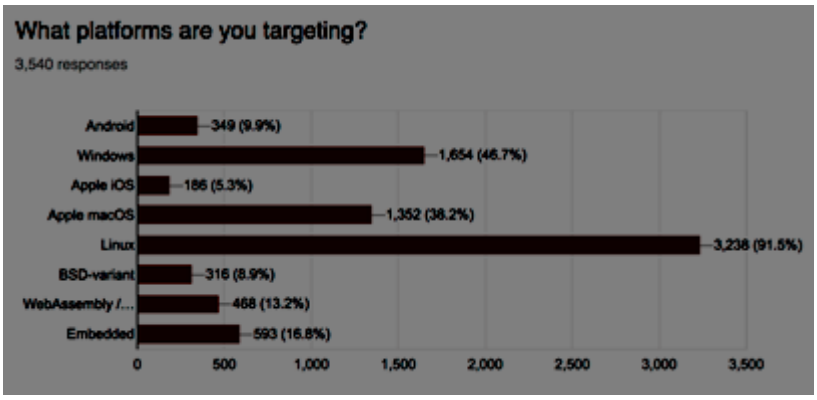
- 人们对于Rust的兴趣与日俱增，相比于2016年，对Rust感兴趣的人增加了约40%,并且呈现出向更多平台发展的趋势。除了在传统的linux系统体系下开发外，也有许多程序员开始在移动端、网

络、嵌入式系统上进行相关的开发。

- 大型程序开发数目的增加。在Rust的用户群中，越来越多的人倾向于使用Rust来编写大型的程序，实现更加复杂的功能。
- 程序的多样化正在逐步增加。调查报告显示，利用Rust开发的程序的种类正在逐步增加。
- Rust的程序的稳定性得到了很好的改善，2016年，有16.2%的用户反映编译器的升级会导致代码崩溃，然而2017年，这一数字下降到了7.5%，表明Rust的整体稳定性有着稳步的提升。

但是，调查报告同时也显示了Rust存在的一些问题：

- Rust上手较为困难，23%的用户反映Rust难以使用，其语法难以理解，编程难度较大;另外还有5%的用户反映Rust缺乏更好的IDE支持。
- Rust的函数库仍需改进，并且编译速度较慢，缺乏相应的工具支持。



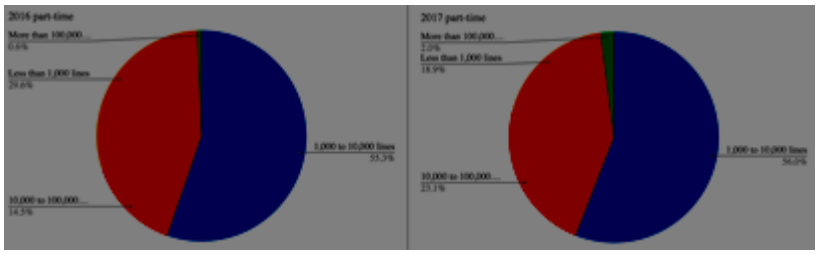
图：利用Rust软件开发的目标平台

4.相关工作

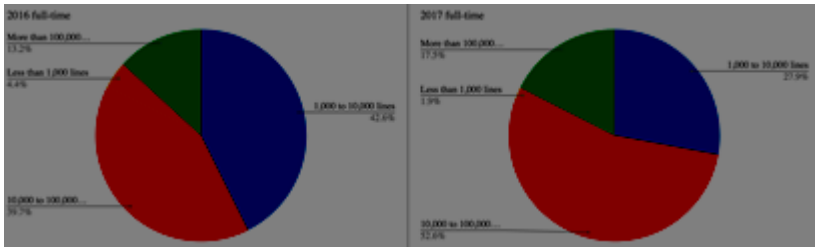
4.1 Rust 应用：

虽然从使用频率等方面比较，Rust仍然无法和c语言等主流语言比肩，但目前来看，Rust也已经在各类企业软件中得到了较为广泛的运用，从官网的调查数 据显示，相比与2016年，2017年中在工作中使用Rust编程的比例有着相当大的提升，并且向着大规模方向发展。[12]

图：2016年和2017年完全利用Rust的开发项目规模对比



图：2016年和2017年部分利用Rust的开发项目规模对比



4.1.1 firefox

随着Rust语言的逐渐成熟，其开发者Mozilla也逐步将其推向实际应用。目前，Mozilla开始尝试用Rust开发Firefox的组件，其中包 括核心引擎Servo/Project Quantum，这一举动将Rust和firefox的整体命运捆绑在了一起，Rust 程序代码也正式呈现给了数亿名 Firefox 用户。Mozilla 初步测得 Rust 组件的执行效果，其结果与原始的 C++ 组件不相上下，而且现在的内存程序语言更安全。

4.1.2 Servo project

利用Rust来编写浏览器的内核程序有着极高的安全性。目前的浏览器，如IE、Chrome等，都采用C++来编写其核心引擎。其原因是C++可以让系统程序员对底层的硬件特征和内存使用有直接的控制，同时还提供一个完全透明的编译模型。

在此基础上，Mozilla开始了名为Servo的项目，旨在建立起一个全新的浏览器，在保留原有的浏览器引擎的功能的前提下，利用最近的并行硬件趋势， 使得浏览器的内存使用更为安全。因此，团队采用了Rust编程语言，保证了C++对于底层系统的控制，同时形成了一个便于使用的、安全的浏览器内核。

在Macbook Pro上测试的结果显示，尽管用Rust编写的内核提供了额外的内存安全保护，但是其运行速度仍然不逊于目前通用的浏览器。现在，Servo引擎已经搭载在了Firefox中，成为了Firefox的新内核。[14]

Site	Gecko	Servo 1 thread	Servo 4 threads
Reddit	250	100	55
CNN	105	50	35

Table 1. Performance of Servo against Mozilla's Gecko rendering engine on the layout portion of some common sites. Times are in milliseconds, where lower numbers are better.

图：Servo的效果及其与传统浏览器内核的对比

4.1.3 TIKV

TiKV 是一个事务型分布式 Key-Value 数据库，是作为 TiDB 项目的核心存储组件，也是 Google 著名的分布式数据库 Spanner 的开源实现。基于Rust的安全性和高性能等诸多优点，这样一个大型的分布式存储项目选择了 Rust 语言来构建。不使用传统的编程软件的主要原因，一是如Java 和 Go 这类编程语言的主要问题是会因 GC 引起抖动，尤其在读写压力比较大的情况下，更容易产生此类问题，二是像C++一类的编程语言，虽然从原理上讲非常适合TIKV的构建，但是由于其自身的特 点，安全性较差，而且一旦出现bug，其维护成本相当巨大。而正如前面提到的，Rust具备很高的安全性，适合这一类数据库的编程和构建。[13]

4.1.4 Redox

Redox 是采用 纯Rust 编程语言而开发的一套操作系统，这是一个发布在GitHub上的开源项目，它的目标是提供一个安全而免费的类Unix微内核操作系统。其主要特点是微内核 (Microkernel)设计、具有图形化的用户界面 Orbital、以及驱动程序在用户空间执行等。

Redox 包含常见的 Unix 命令，为 C 程序提供了 Newlib 库，提供 ZFS 文件系统支持。除此之外，Redox 还有 shell 程序 Ion、包管理器 Magnet、以及类似 vi 的文本编辑器 Sodium 等应用。

Rust 这门新兴的语言对于国内大多数开发者来说会显得比较陌生，但是并不妨碍 Rust 已经在世界范围内作为公认的 C/C++ 的有希望的挑战者。从长远来看，在对内存安全性和性能有严苛要求的场景，Rust 将会有广阔的发展空间。

4.2 FreeRTOS国际上应用现状，待解决的问题

目前，RTOS在国际市场上有着举足轻重的地位。 RTOS 已经在全球形成了一个产业，据美国EMF(电子市场分析)报告，1999年全球RTOS市场产值达3.6亿美圆，而相关的整个嵌入式开发工具（包括仿真器、逻辑分析仪、软件编译器和调试器）则高达9亿美元。

4.2.1 RTOS发展历史

从1981年Ready System发展了世界上第1个商业嵌入式实时内核（VRTX32），到今天已经有近40年的历史。20世纪80年代的产品还只支持一些16位的微处理 器，如68k,8086等。这时候的RTOS还只有内核，以销售二进制代码为主。当时的产品除VRTX外，还有IPI公司的MTOS和80年代末ISI公 司的PSOS。产品主要用于军事和电信设备。进入20世纪90年代，现代操作系统的设计思想，如微内核设计技术和模块化设计思想，开始渗入RTOS领域。 老牌的RTOS厂家如Ready System（在1995年与Microtec Research合并），也推出新一代的VRTXsa实时内核，新一代的RTOS厂家Windriver推出了Vxwork。另外在这个时期，各家公司都 有力求摆脱完全依赖第三方工具的制约，而通过自己收购、授权或使用免费工具链的方式，组成1套完整的开发环境。例如，ISI公司的Prismt、著名的Tornado(Windriver)和老牌Spectra(VRTX开发系统)等。

到了20世纪90年代中期，互联网之风在北美日渐风行。网络设备制造商、终端产品制造商都要求RTOS有网络和图形界面的功能。为了方便使用大量现存的软 件代码，他们希望RTOS厂家都支持标准的API，如POSIX， Win32等，并希望RTOS的开发环境与他们已经熟悉的UNIX，Windows一致。这个时期代表性的产品有Vxwork，QNX,Lynx和 WinCE等。

4.2.2 RTOS市场和技术发展的变化

可以看出，20世纪90年代后，RTOS在嵌入式系统设计中的主导地位已经确定，越来越多的工程师使用RTOS，更多的新用户愿意选择购买产品而不是自己开发。RTOS的技术发展有以下一些变化：

1. 因为新的处理器越来越多，RTOS自身结构的设计更易于移植，以便在短时间内支持更多种微处理器。
2. RTOS厂家逐渐走上开放源码的道路。相当多的RTOS厂家出售RTOS时，就附加了源程序代码并含生产版税。
3. 后PC时代更多的产品使用RTOS，如手持设备等，它们对实时性要求并不高。微软公司的WinCE,Plam OS, Java OS等ＲＴＯＳ产品就是顺应这些应用而开发出来的。

4. 电信设备、控制系统要求的高可靠性，对ＲＴＯＳ提出了新的要求。瑞典Enea公司的OES和WindRiver 新推出的Vxwork ＡＥ对支持ＨＡ（高可用性）和热切换等特点都下了一番功夫。
5. Windriver收购了ISI，在RTOS市场形成了相当程度的垄断，但是由于Windriver决定放弃 PSOS，转为开发Vxwork与PSOS合二为一版本，这便使得PSOS用户再一次走到重新选择RTOS的路口，给了其他RTOS厂家一次机会。
6. 嵌入式Linux已经在消费电子设备中得到应用。韩国和日本的一些企业都推出了基于嵌入式Linux的手持设备。嵌入式Linux得到了相当广泛的半导体厂商的支持和投资，如Intel和 Motorola。

4.2.3RTOS的未来

未来RTOS的应用可划分为3个不同的领域：

1. 系统级：指RTOS运行在1个小型的计算机系统中完成实时的控制作用。这个领域将主要是微软与Sun竞争之地，传统上Unix在这里占有绝对优 势。Sun通过收购，让他的Solaris与 Chrous os（原欧洲的1种RTOS）结合，微软力推NT的嵌入式版本“Embedded NT”。 此外，嵌入式Linux将依托源程序码开放和软件资源丰富的优势，进入系统级RTOS的市场。
2. 板级：传统的RTOS的主要市场。如Vxwork, PSOS, QNX, Lynx和VRTX，其应用将主要集中在航空航天、电话电讯等设备上。
3. SoC级：新一代RTOS的领域：主要应用在消费电子、互联网络和手持设备等产品上。代表的产品有Symbian 的Epoc、ATI 的Nucleus, Express logic 的Threadx。老牌的RTOS厂家的产品VRTX和Vxwork也很注意这个市场。

从某种程度讲，不会出现一个标准的RTOS（像微软的Windows在桌面系统中的地位一样），因为嵌入式应用本身就极具多样性。但是，在某个时间段以及 某种行业，会出现一种绝对领导地位的RTOS，比如今天在宽带的数据通信设备中的Vxwork和在亚洲手持设备市场上的WinCE就是一例子。但是，这种 垄断地位也并不是牢不可破的，因为在某种程度上用户和合作伙伴更愿意去培养1个新的竞争对手。比如，Intel投资的Montivista和 Motorola投资的Lineo，这两家嵌入式Linux系统，就是说明半导体厂商更愿意看到1个经济适用的、开放的RTOS环境。

4.2.4RTOS在中国

中国将是世界上最大的RTOS市场之一。因为中国有着世界上最大的电信市场。据信息产业部预计，在未来2~3年内，中国将是世界上最大的手机市场（每1部 手机都在运行1个RTOS）。这样庞大的电信市场孕育着大量的电信设备制造商，这就造就了大量的RTOS，也提供给开发工具市场巨大的机会。目前，中国的 大多数设备制造商在采用RTOS时，首先考虑的还是国外产品。目前，在中国市场上流行的RTOS主要有Vxwork,PSOS, VRTX, Nucleus，QNX和WinCE等。由于多数RTOS是嵌入在设备的控制器上，所以多数用户并不愿意冒风险尝试一种新的RTOS。

但是，目前 RTOS在中国市场的销售额还很小，这主要是2个原因：

1. 中国设备制造商的规模普遍还无法与国外公司相比，开发和人员费用相对还较高，所以RTOS对于中国用户来讲是比较昂贵的。
2. 多数国内用户还没有开始购买RTOS的版权，其主要原因是产品未能批量生产，用户没有交纳版税的意识。

在过去的几年中，国家研究机构和企业已经在开发自有知识产权的RTOS或在开放源码的Linux基础上发展自己的嵌人式Linux版本。国产RTOS的市 场主要集中在消费电子方面，因为该领域有许多国外RTOS不能适应的部分，如中文处理。目前主要产品有：中科院系统的“女娲”，北京科银京成的δ OS，中科院红旗Linux，深圳蓝点Linux。可以肯定地讲，目前这些RTOS市场占有率还很低，多数公司还是依靠政策支持、国内投资、海外上市等支 持公司庞大的开发投入，真正的市场回报还只是杯水车薪。如何长期良性循环发展下去将是一个重要的课题。

如何开发出一种通用的RTOS，使得用户易于使用，方便地裁剪到某系统中去，国外商用RTOS已经很好地解决了这个问题。中国人设计的RTOS应更多地适 于中国的国情，除了中文处理，中国有着广泛的单片机的应用基础。开发设计一种简单、易用的RTOS开发环境，以中国人可以接受的价格和更为务实的技术支持 手段推出，也许可以找到一种正常的市场回报途径。RTOS产业是一个循序渐进的产业，任何急功近利的做法都将导致功亏一篑。用户熟悉一种RTOS需要一个 相当的过程和厂家的支持，同时用户也不愿意轻易放弃一种RTOS。我们相信中国人自己开发设计的RTOS一定会得到国人的认可，有着无限光明的前途。

4.2.5 FreeRTOS与uC/OS II两种嵌入式实时操作系统的比较

μC/OS-II由Micrium公司提供，是一个可移植、可固化的、可裁剪的、占先式多任务实时内核，它适用于多种微处理器，微控制器和数字处理芯片（已经移植到超过100种以上的微处理器应用中）。同时，该系统源代码开放、整洁、一致，注释详尽，适合系统开发。

freeRTOS比uC/OS II优胜的地方：

1. 内核ROM和耗费RAM都比uC/OS 小，特别是RAM。 这在单片机里面是稀缺资源，uC/OS至少要5K以上， 而freeOS用2~3K也可以跑的很好。
2. freeRTOS 可以用协程（Co-routine），减少RAM消耗（共用STACK）。uC/OS只能用任务（TASK，每个任务有一个独立的STACK）。
3. freeRTOS 可以有优先度一样的任务，这些任务是按时间片来轮流处理，uCOSII 每个任务都只有一个独一无二的优先级。因此，理论上讲，freeRTOS 可以管理超过64个任务，而uC/OS只能管理64个。
4. freeRTOS 是在商业上免费应用。uC/OS在商业上的应用是要付钱的。

freeRTOS 不如uCOS的地方：

1. 比uC/OS简单，任务间通讯freeRTOS只支持Queque， Semaphores， Mutex。 uCOS除这些外，还支持Flag, MailBox.
2. uC/OS的支持比freeRTOS 多。除操作系统外，freeRTOS只支持TCPIP， uC/OS则有大量外延支持，比如FS， USB， GUI， CAN等的支持
3. uC/OS可靠性更高，而且耐优化，freeRTOS 在我设置成中等优化的时候，就会出问题。

5.总结

在调研工作中，我们了解了嵌入式系统的安全性问题。在未来，随着物联网的发展，嵌入式系统将得到更加广泛的应用，因而它的安全性必须得到进一步的加强。同 时，我们发现，目前大部分的操作系统内核都是由C/C++编写的，虽然它们有着很高的运行效率，但存在着天生的安全性不足，编写新

的内核，需要更加强有力 的工具。

Rust语言是一门新生的语言，年轻而富有活力。虽然只是一门小众语言，但它的优良特性——安全性、高效性、并发性特别适合用于系统的开发，是取代C/C++的良好选择。目前，越来越多的项目考虑采用Rust编写。在可见的将来，Rust将会得到更加广泛的应用。

根据上面的调研工作，我们准备使用Rust语言重写一个嵌入式内核。我们预计，新的开发工具能显著提升嵌入式内核的安全性，并可能带来效率的提升。总之，这是一个有益的尝试，也与业界的发展方向一致，具有很高的实际价值。

6.参考资料

[1].《嵌入式操作系统风云录：历史演进与物联网未来》 **何小庆**

[2].《Embedded Systems Dictionary》**Jack Ganssle, Michael Barr**

[3].《嵌入式系统的定义与发展历史》 **何立民**

[4]. 浅谈RAII惯用法 **张浩,杨杰**

[5]. Rust语言：安全地并发 **孙宁**

[6] Evaluation of performance and productivity metrics of potential programming languages in the HPC environment **Florian Wilkens**

[7]. Lock-freedom without garbage collection-Aaron Tu

[8].《嵌入式系统的实时性问题》，北京航空航天大学 **何立民**

[9].《嵌入式Linux系统安全性探析》**王友顺，曲豪**

[10].《嵌入式系统:从SoC芯片到系统/凌明, 王学香, 单伟伟》 电子工业出版社

[11]. Three-Dimensional Integrated Circuits and the Future of System-on-Chip Designs **R.S. Patti**

[12]. 2017Rust调查报告

[13]. 我们为什么要选择小众语言Rust来实现TiKV?

[14]. Experience Report: Developing the Servo Web Browser Engine using Rust **Brian Anderson, Lars Bergstrom, Manish Goregaokar**

[15].《嵌入式实时操作系统的现状和未来》

[16].《2018年工业机器人行业现状与发展趋势分析》