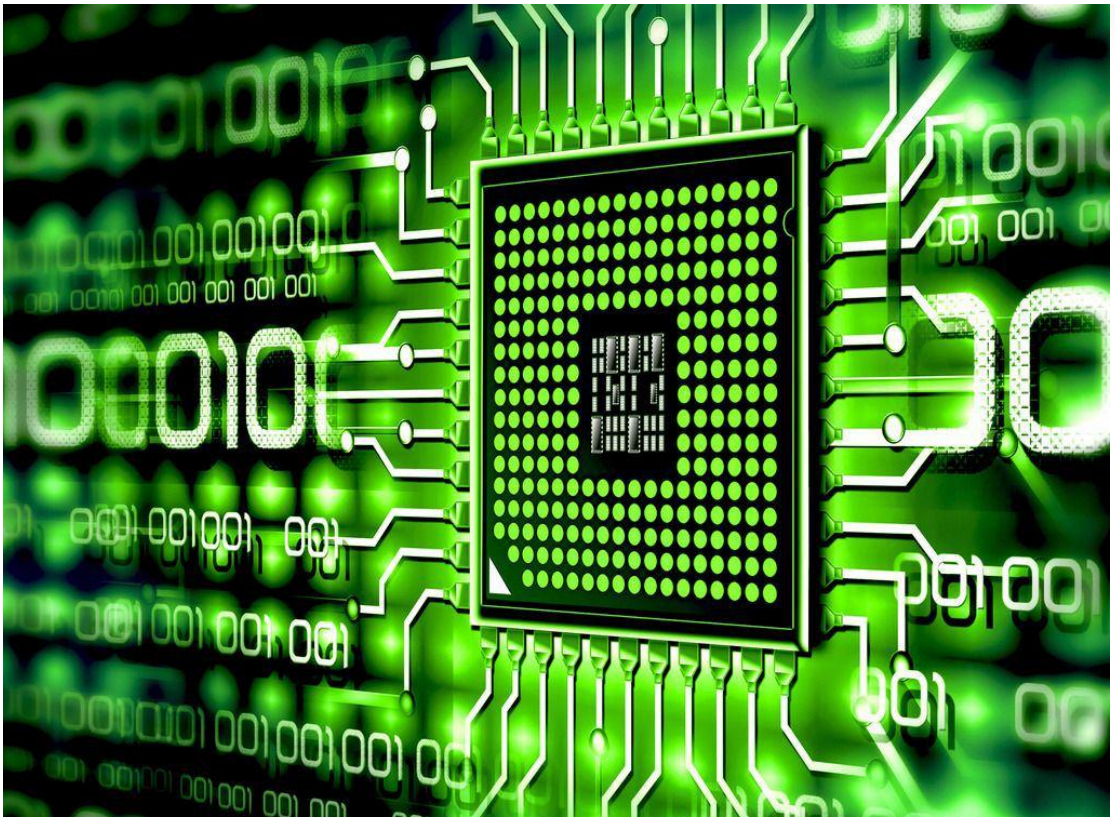


# 关于中国市场汽车智能驾驶主控芯片操作系统的思考

焉知汽车
已认证账号



**作者** / 孙鲁毅（安霸半导体上海软件研发高级总监）

关于中国的汽车智能驾驶市场，针对不同产品该采用什么操作系统呢？相信来自汽车行业的开发团队都有自己的观点。作为在芯片原厂的系统开发团队，我们经历了多款安霸半导体(Ambarella)的视频和视觉芯片的系统开发实践，并参与了国内国外诸多客户的项目成功量产，这些积累使得我们对不同操作系统的性能特点和差别得到一定的了解。在汽车行业努力耕耘的几年里，我们也发现了汽车市场对于芯片和操作系统需求的不同，因此积极努力，不断推陈出新，争取给客户和市场带来更好的体验。以下是有关智能驾驶主控芯片的操作系统的研究。

## 一．常见操作系统的对比

据我们观察，汽车市场的操作系统选择主要来自于开发团队的经验积累，或者来自于主控芯片厂的SDK的默认支持。按其特性分为两类：一个是各种各样的Linux(包含标准Linux, Android, AGL，商用Linux等)，一个是各种各样的RTOS （实时操作系统, 包含QNX，GHS，Vxworks, ThreadX，Nucleus，FreeRTOS，µITRON，厂商自制的RTOS， RT-Thread等），而RTOS又可根据特性分为简单RTOS和高级RTOS两类。

关于这几种OS大体上列表比较如下：

| | | | |
|---|---|---|---|
| 常见车载操作系统的三大类 | 传统的简单RTOS（如ThreadX, Nucleus，FreeRTOS，µITRON, 厂商自研的RTOS等） | 兼容POSIX，生态丰富的的高级RTOS，如QNX等 | Linux内核操作系统（AGL, Android ,厂商维护的Linux，商用Linux等） |

| | | | |
|---|---|---|---|
| 支持CPU种类 | 各种常见的MCU，ARM等指令集的SOC，大多为32-bit，有的可以支持多核CPU | 支持内存保护的CPU比如ARM,可很好支持32-bit/64-bit，很好地支持多核 | 支持内存保护的CPU比如ARM，可很好支持32-bit/64-bit，很好地支持多核 |
| 实时性 | 很好，微秒级任务调度延迟 | 很好，微秒级任务调度延迟 | 毫秒级（不打开CONFIG_PREEMPT_RT）；微秒级（打开CONFIG_PREEMPT_RT） |
| 启动速度 | 几百毫秒或更短 | 几百毫秒 | 一秒到十秒（使用不同配置和不同的初始化过程） |
| 内核+驱动程序+文件系统典型大小 | 几十到几百KB | 几百KB | 几MB或更大 |
| 支持多线程多任务 | 是 | 是 | 是 |
| 支持多进程和内存保护 | 一般没有 | 是 | 是 |
| 兼容POSIX API | 不支持或非常有限 | 兼容 | 兼容 |
| 开发工具和使用费用 | 很低或免费 | 商业软件，费用较高 | 免费 |
| 软件生态丰富程度 | 较丰富，多为IoT设备，MCU等 | 较丰富，从商业公司提供，或自己从开源软件移植。 | 非常丰富。大量的软件可从开源社区获得，从嵌入式应用到网络，音视频，人工智能等。 |
| 内核功能安全等级 | 常常有ASIL或类似的功能安全等级 | 可达ASIL-D | 没有功能安全等级 |
| 常用于哪些设备 | 仪表盘，IoT设备，行车记录仪，ADAS设备 | 仪表盘，中控娱乐，导航，汽车智能驾驶域控制器等 | 中控娱乐，导航，行车记录仪，T-BOX ADAS设备，某些汽车智能驾驶域控制器等 |

| | | | |
|---|---|---|---|
| 常见车载操作系统的三大类 | 传统的简单RTOS（如ThreadX, Nucleus，FreeRTOS，µITRON，厂商自研的RTOS等） | 兼容POSIX，生态丰富的的高级RTOS，如QNX等 | Linux内核操作系统（AGL, Android ,厂商维护的Linux，商用Linux等） |
| 支持CPU种类 | 各种常见的MCU，ARM等指令集的SOC，大多为32-bit，有的可以支持多核CPU | 支持内存保护的CPU比如ARM,可很好支持32-bit/64-bit，很好地支持多核 | 支持内存保护的CPU比如ARM，可很好支持32-bit/64-bit，很好地支持多核 |
| 实时性 | 很好，微秒级任务调度延迟 | 很好，微秒级任务调度延迟 | 毫秒级（不打开CONFIG_PREEMPT_RT）；微秒级（打开CONFIG_PREEMPT_RT） |
| 启动速度 | 几百毫秒或更短 | 几百毫秒 | 一秒到十秒（使用不同配置和不同的初始 |

| | | | |
|---|---|---|---|
| | | | 化过程） |
| 内核+驱动程序+文件系统典型大小 | 几十到几百KB | 几百KB | 几MB或更大 |
| 支持多线程多任务 | 是 | 是 | 是 |
| 支持多进程和内存保护 | 一般没有 | 是 | 是 |
| 兼容POSIX API | 不支持或非常有限 | 兼容 | 兼容 |
| 开发工具和使用费用 | 很低或免费 | 商业软件，费用较高 | 免费 |
| 软件生态丰富程度 | 较丰富，多为IoT设备，MCU等 | 较丰富，从商业公司提供，或自己从开源软件移植。 | 非常丰富。大量的软件可从开源社区获得，从嵌入式应用到网络，音视频，人工智能等。 |
| 内核功能安全等级 | 常常有ASIL或类似的功能安全等级 | 可达ASIL-D | 没有功能安全等级 |
| 常用于哪些设备 | 仪表盘，IoT设备，行车记录仪，ADAS设备 | 仪表盘，中控娱乐，导航，汽车智能驾驶域控制器等 | 中控娱乐，导航，行车记录仪，T-BOX ADAS设备，某些汽车智能驾驶域控制器等 |

μITRON

| | | | |
|---|---|---|---|
| 常见车载操作系统的三大类 | 传统的简单RTOS（如ThreadX, Nucleus，FreeRTOS，μITRON, 厂商自研的RTOS等） | 兼容POSIX，生态丰富的的高级RTOS，如QNX等 | Linux内核操作系统（AGL, Android ,厂商维护的Linux，商用Linux等） |
| 支持CPU种类 | 各种常见的MCU，ARM等指令集的SOC，大多为32-bit，有的可以支持多核CPU | 支持内存保护的CPU比如ARM,可很好支持32-bit/64-bit，很好地支持多核 | 支持内存保护的CPU比如ARM，可很好支持32-bit/64-bit，很好地支持多核 |
| 实时性 | 很好，微秒级任务调度延迟 | 很好，微秒级任务调度延迟 | 毫秒级（不打开CONFIG_PREEMPT_RT）；微秒级（打开CONFIG_PREEMPT_RT） |
| 启动速度 | 几百毫秒或更短 | 几百毫秒 | 一秒到十秒（使用不同配置和不同的初始化过程） |
| 内核+驱动程序+文件系统典型大小 | 几十到几百KB | 几百KB | 几MB或更大 |
| 支持多线程多任务 | 是 | 是 | 是 |
| 支持多进程和内存保护 | 一般没有 | 是 | 是 |
| 兼容POSIX API | 不支持或非常有限 | 兼容 | 兼容 |
| 开发工具和使用费用 | 很低或免费 | 商业软件，费用较高 | 免费 |
| 软件生态丰富程度 | 较丰富，多为IoT设备，MCU等 | 较丰富，从商业公司提供，或自己从开源软件移植。 | 非常丰富。大量的软件可从开源社区获得，从嵌入式应用到 |

| | | | |
|---|---|---|---|
| | | | 网络，音视频，人工智能等。 |
| 内核功能安全等级 | 常常有ASIL或类似的功能安全等级 | 可达ASIL-D | 没有功能安全等级 |
| 常用于哪些设备 | 仪表盘，IoT设备，行车记录仪，ADAS设备 | 仪表盘，中控娱乐，导航，汽车智能驾驶域控制器等 | 中控娱乐，导航，行车记录仪，T-BOX ADAS设备，某些汽车智能驾驶域控制器等 |

| | | | |
|---|---|---|---|
| 常见车载操作系统的三大类 | 传统的简单RTOS（如ThreadX, Nucleus，FreeRTOS，µITRON, 厂商自研的RTOS等） | 兼容POSIX，生态丰富的的高级RTOS，如QNX等 | Linux内核操作系统（AGL, Android ,厂商维护的Linux，商用Linux等） |
| 支持CPU种类 | 各种常见的MCU，ARM等指令集的SOC，大多为32-bit，有的可以支持多核CPU | 支持内存保护的CPU比如ARM,可很好支持32-bit/64-bit，很好地支持多核 | 支持内存保护的CPU比如ARM，可很好支持32-bit/64-bit，很好地支持多核 |
| 实时性 | 很好，微秒级任务调度延迟 | 很好，微秒级任务调度延迟 | 毫秒级（不打开CONFIG_PREEMPT_RT）；微秒级（打开CONFIG_PREEMPT_RT） |
| 启动速度 | 几百毫秒或更短 | 几百毫秒 | 一秒到十秒（使用不同配置和不同的初始化过程） |
| 内核+驱动程序+文件系统典型大小 | 几十到几百KB | 几百KB | 几MB或更大 |
| 支持多线程多任务 | 是 | 是 | 是 |
| 支持多进程和内存保护 | 一般没有 | 是 | 是 |
| 兼容POSIX API | 不支持或非常有限 | 兼容 | 兼容 |
| 开发工具和使用费用 | 很低或免费 | 商业软件，费用较高 | 免费 |
| 软件生态丰富程度 | 较丰富，多为IoT设备，MCU等 | 较丰富，从商业公司提供，或自己从开源软件移植。 | 非常丰富。大量的软件可从开源社区获得，从嵌入式应用到网络，音视频，人工智能等。 |
| 内核功能安全等级 | 常常有ASIL或类似的功能安全等级 | 可达ASIL-D | 没有功能安全等级 |
| 常用于哪些设备 | 仪表盘，IoT设备，行车记录仪，ADAS设备 | 仪表盘，中控娱乐，导航，汽车智能驾驶域控制器等 | 中控娱乐，导航，行车记录仪，T-BOX ADAS设备，某些汽车智能驾驶域控制器等 |

我们了解到在某些领域，比如Android凭借在平板和手机领域的地位，在国产新能源车的中控娱乐系统上占了很大的市场份额， 而BBA的新款车中控则采用了QNX，其它一些车型采用的是AGL。如果我们来看ADAS市场和自动驾驶市场，早期采用雷达和相对简单的传统视觉算法时， RTOS搭配

经典AUTOSAR的使用比较普遍，而在近些年来的各种新涌现的基于神经网络算法的ADAS解决方案里Linux占比开始增加。在APA辅助泊车领域里，多路摄像头+ GPU的应用搭配Linux/Android更加流行。 在自动驾驶时代，QNX和Linux都成为比较流行的自动驾驶域控制器的OS选择。众所周知，特斯拉采用了自己维护的Linux系统做Autopilot并取得广泛好评。

传统意义上，Linux被认为是实时性不够好，而RTOS如其名Real-time OS， 可以保证任务调度（ISR，线程切换等）总是可以在几个微秒之内调度，除非驱动程序有错误卡死。所有RTOS上的任务严格按照优先级，高优先级可以抢占低优先任务，同优先级的任务一般按轮转调度策略，一切井然有序。目前基于雷达的ADAS产品一般都采用RTOS，就是利用其硬实时性带来的安全保障，而且产品开发阶段需求都已经非常清晰，所以可以给每个人任务固定的优先级。 而随着Linux内核技术的发展，如今的Linux的实时性已经大为改观，具体有哪些改观呢？请参见后面章节详述。

就内核大小而言，RTOS一般都更有优势，更小，加载更快，通常更有利于在简单处理器上执行。但因为Linux内核是高度可配置的，而且驱动便于随时加载卸载。经过少许配置和优化，可使标准Linux在1秒左右的时间在主流ARM CPU完成冷启动到shell界面。

而高级RTOS比如QNX, GHS在应用程序层面，和Linux一样开发都比较便利，而且可以使用POSIX API，便于移植。所有高级RTOS都是商用OS，你付费购买开发的license，就可以得到相应的服务，而且比起随便找一个Linux版本测试的话，更加成熟稳定。 比如QNX是基于微内核架构的，具有内存保护，支持应用程序动态加载，支持各种常见驱动和文件系统和网络协议，包含对多种嵌入式芯片的支持。

而如果你使用Linux，则可以尝试Linux家族中采用Android, AGL，采用芯片开发商的定制Linux或者自行配置内核。大部分不同的Linux的内核都一样，只是用了不同的配置选项。Linux内核特别针对ARM架构的各种CPU的支持已经非常成熟，而且驱动程序框架也很完整，用户空间的各种软件生态非常丰富。如果和QNX相比，Linux的突出特点在于免费，给用户更大的灵活性，更多的应用场景，以及更为丰富的软件库选择。而QNX内核则具有更高的功能安全等级，以及商业软件和工具带来的保障和服务。

## 二． Linux和RTOS的实时性对比

Linux常被工业界诟病为“实时性”不佳，任务调度延时不确定，是这样吗？

对于大学本科或者研究生里开设“操作系统”课程而言，一般都称Linux为一种典型的“非实时系统”，以有别于RTOS (Real-Time Operating System)。 原因就是RTOS严格根据优先级决定调度，高优先级的任务可以抢占低优先级的任务；但Linux的调度机制较为复杂，是以完全公平调度（CFS）为基础的，支持内核抢占式调度，以及实时线程的一种综合机制。

那么我们要问，这个任务调度延时是多长呢？ 一般说来，硬实时的RTOS需要这个调度延时最坏情况在10微秒到100微秒左右。而软实时的桌面式操作系统则最坏情况在1~10毫秒左右。

Linux 1.0开始继承了分时调度机制的粗时间粒度，和相对差的实时性，但对于Linux的实时性改进的工作从未停止过。2005年，Linux2.6的发布，就开始支持内核可抢占式调度CONFIG_PREEMPT，此选项使得内核态程序只要不被spinlock保护或者在中断处理程序里，都可被更高优先级的内核线程抢占。这就使得很多任务执行的时候，任务调度延迟大大改善。默认配置通常可以做到调度延迟在1到几个毫秒。这以后各种实时性优化补丁被开发出来，比如带有实时内核外加普通Linux的RTLinux, RTAI,以及直接改善Linux内核实时性的配置选项等，使得Linux的实时性得到不断改善。实时性补丁CONFIG_PREEMPT_RT项目也已经比较成熟，可以手动打patch添加提升内核的实时性。（参加OSADL组织的Realtime Linux项目）

osadl.org/Realtime-Linu...

CONFIG_PREEMPT_RT选项通过修改内核锁，使得Spinlock, rwlock都可重入，并且实现了内核中spinlock和信号量的优先级继承，把中断处理程序都变成了可重入的内核线程，并且用高精度内核定时器替代了传统的时间函数，这些优化大大改善了Linux的实时性，使得Linux实际已经成为硬实时操作系统。

德国不莱梅大学曾做CONFIG_PREEMPT_RT配置下Linux 实时性 vs RTOS的测试。（来源：pdfs.semanticscholar.org...）

结果表明， Linux内核的实时性已经接近RTOS，都可以达到微秒级延时。

| Tasks | Partition | **HRTL** | **RT-Preempt** | **QNX** | Scenario |
|---|---|---|---|---|---|
| 2 | static/none | 0.002 (0.471) | 0.004 (0.617) | | |
| | | 0.002 (0.469) | 0.007 (0.616) | | CPU |
| | | 0.007 (0.474) | 0.004 (0.623) | | I/O |
| 2 | dynamic | 0.008 (0.723) | | 0.176 (0.440) | |
| | | 0.009 (0.737) | | 0.196 (0.463) | CPU |
| | | 0.009 (0.726) | | 0.547 (1.129) | I/O |
| 16 | static/none | 0.014 (0.574) | 0.016 (0.698) | | |
| 16 | dynamic | 0.012 (0.828) | | 0.196 (0.465) | |
| 128 | static/none | 0.030 (0.778) | 0.037 (0.910) | | |
| 128 | dynamic | 0.024 (1.039) | | 0.583 (0.639) | |
| 512 | static/none | 0.109 (1.219) | 0.114 (1.309) | | |
| 512 | dynamic | 0.106 (1.488) | | 2.008 (0.824) | |

**Table 12.4.:** Task switch benchmark results overview [$\mu s$]

采用标准Linux kernel 5.4内核即使不加实时性补丁，仅仅把CONFIG_PREEMPT做成默认，实际也可以达到很好的实时性。对此我们做了长时间压力测试，条件如下：

- CPU为4核ARM Cortex A53，运行于1GHz
- 整个SoC系统做高性能4Kp30视频采集和H.265编码，
- 同时满负荷深度学习计算
- 并加50个busyloop 进程让CPU保持100%忙
- 维持整个测试过程24个小时

结果如下：

```
Kernel 5.4.45:T: 0 ( 384) P:90 I:1000 C:232187688 Min: 4 Act: 8 Avg: 7 Max: 108T: 1 ( 385) P:90 I:1500 C:154791787 Min: 4 Act: 7 Avg: 7 Max: 41T: 2 ( 386) P:90 I:2000 C:116093836 Min: 4 Act: 8 Avg: 6 Max: 26T: 3 ( 387) P:90 I:2500 C:92875065 Min: 4 Act: 8 Avg: 6 Max: 25
```


```
Kernel 5.4.45:T: 0 ( 384) P:90 I:1000 C:232187688 Min: 4 Act: 8 Avg: 7 Max: 108T: 1 ( 385) P:90 I:1500 C:154791787 Min: 4 Act: 7 Avg: 7 Max: 41T: 2 ( 386) P:90 I:2000 C:116093836 Min: 4 Act: 8 Avg: 6 Max: 26T: 3 ( 387) P:90 I:2500 C:92875065 Min: 4 Act: 8 Avg: 6 Max: 25
```

```
Kernel 5.4.45:T: 0 ( 384) P:90 I:1000 C:232187688 Min: 4 Act: 8 Avg: 7 Max: 108T: 1 ( 385) P:90 I:1500 C:154791787 Min: 4 Act: 7 Avg: 7 Max: 41T: 2 ( 386) P:90 I:2000 C:116093836 Min: 4 Act: 8 Avg: 6 Max: 26T: 3 ( 387) P:90 I:2500 C:92875065 Min: 4 Act: 8 Avg: 6 Max: 25
```

```
Kernel 5.4.45:T: 0 ( 384) P:90 I:1000 C:232187688 Min: 4 Act: 8 Avg: 7 Max: 108T: 1 ( 385) P:90 I:1500 C:154791787 Min: 4 Act: 7 Avg: 7 Max: 41T: 2 ( 386) P:90 I:2000 C:116093836 Min: 4 Act: 8 Avg: 6 Max: 26T: 3 ( 387) P:90 I:2500 C:92875065 Min: 4 Act: 8 Avg: 6 Max: 25
```


```
Kernel 5.4.45:T: 0 ( 384) P:90 I:1000 C:232187688 Min: 4 Act: 8 Avg: 7 Max: 108T: 1 ( 385) P:90 I:1500 C:154791787 Min: 4 Act: 7 Avg: 7 Max: 41T: 2 ( 386) P:90 I:2000 C:116093836 Min: 4 Act: 8 Avg: 6 Max: 26T: 3 ( 387) P:90 I:2500 C:92875065 Min: 4 Act: 8 Avg: 6 Max: 25
```



```
Kernel 5.4.45:T: 0 ( 384) P:90 I:1000 C:232187688 Min: 4 Act: 8 Avg: 7 Max: 108T: 1 ( 385) P:90 I:1500 C:154791787 Min: 4 Act: 7 Avg: 7 Max: 41T: 2 ( 386) P:90 I:2000 C:116093836 Min: 4 Act: 8 Avg: 6 Max: 26T: 3 ( 387) P:90 I:2500 C:92875065 Min: 4 Act: 8 Avg: 6 Max: 25
```


这项测试结果表明Linux调度延迟最坏情况不超过108微秒，调度延迟平均值不超过8微秒。这项测试表明在系统满负荷长时间运转时，Linux标准内核不加任何patch也不亚于常见各种实时操作系统的实时性。

（此项测试使用OSADL的SIL2LinuxMP项目的实时性测试工具cyclictest完成。

osadl.org/)

那我们要问，既然Linux实时性变得这么好了，RTOS的存在意义在哪里呢？其实也不尽然。因为Linux虽然可以创建实时线程（和实时进程），但本质上的调度并不是严格按照优先级调度。如果你需要严格按照特定的顺序调度程序，在Linux里的开发方法可以使用线程同步，或者把任务装进一个队列等机制，而不是依靠绝对优先级的高低。相对来说，RTOS的方式更为简单和确定，但Linux的方式更灵活也更利于系统任务的扩展和承载更大的吞吐量。

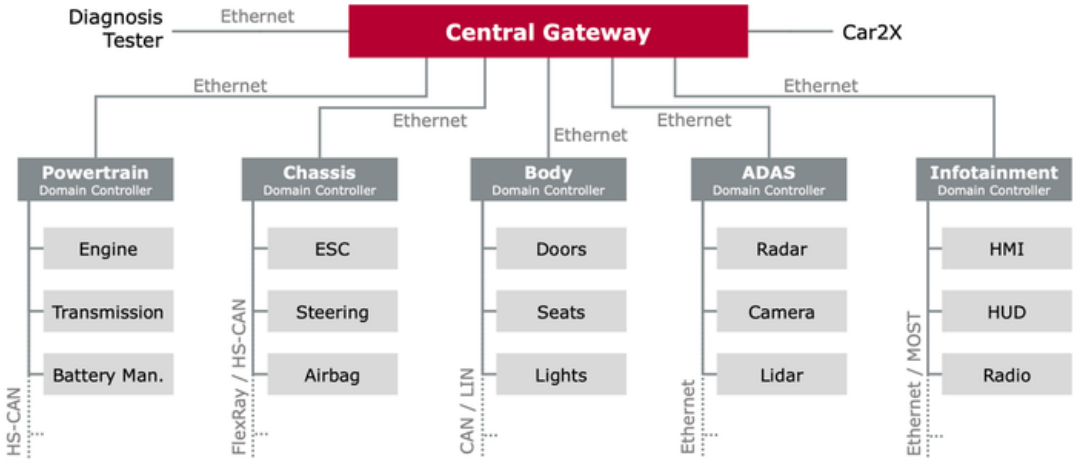
不管Linux还是RTOS，实时性都和CPU收到中断的频率，中断处理程序占用的时间等有关。先进的处理器支持一些硬件机制可以减小关中断的时间。在Linux里，也可以把时间要求最苛刻的任务放在中断处理程序里做或者做一部分，所以在现代CPU里搭配新的Linux内核以后，只要按照合理的编程规范和做必要的代码检查，实时性接近RTOS是有保证的。但因为Linux的调度机制非常复杂，所以至少无法做得像RTOS调度器那样“简单易懂”，如果是一个对实时性和确定性能要求极高的场合，比如汽车上控制刹车的电子稳定控制系统的ECU使用Linux是不太可能的。 但是如果对于摄像机而言，Linux的实时性就已经远远好于系统所需要的指标。对比一下，人眼的视觉暂留也是一种神经传输延迟，高达100毫秒以上。

## 三．系统架构和硬件对操作系统的影响

早期的汽车电子电气架构，是分立式ECU,各自独立存在，为不同功能独立设计的系统。后来的主流架构演变成为以CAN总线或其他总线为核心，多个ECU相互合作，并划分为多个域的系统，从而大大降低总线束长度和系统复杂性。

而未来这种系统架构，受到自动驾驶技术的驱动，可能在每个域内采用一个或几个域控制器来代替多个分立的ECU，并且数据总线将会变得更加高速和简洁，如千兆/万兆以太网作为数据传输的骨干网络，以便接入大量的6个以上的摄像头作为视觉感知，以及各种雷达和其他传感器。

简单的说,汽车架构将会是一个包含多个网关的“局域网”，内部包含多个传感器，控制器，和执行器。有的简单传感器和执行器则更看重功能安全，实现简单，以及重用已经有的设计，对这些场景来说，继续沿用本来的RTOS设计是非常合理的。传感器和视觉相关的就是多个摄像头，控制器和算法相关如舱内域控制器，自动驾驶域控制器等。而摄像头和域控制器的复杂度已经远远超过了早期的ECU，而变成了嵌入式计算机系统，这就产生了对于操作系统功能更多的要求。




（图片来自于Vector公司的“TC8 conformance Testing of Automotive Ethernet Networks”）

正因为系统复杂度越来越大，软件的创新也越来越多，和新车交付时间从设计到上市越来越短，这使得系统中的崭新功能模块不一定能够在第一个版本就已经完善，所以系统在线升级（OTA）成为必须。几乎所有车载嵌入式软件系统，都可以通过OTA升级。这使得通用的架构设计，网络协议，计算机安全等技术环节对汽车系统比以往都更加重要。

这种架构的升级，使得系统需要使用高级的CPU加上高级的操作系统，才有利于软件的移植，才可以利用丰富的软件库，和成熟的软件开发工具，丰富的文档，或者直接开放源代码。简单MCU加上简单RTOS已经很难适应这种需求。

针对这些新需求，为了改善系统的实时性和可靠性，硬件也在快速发展并使得操作系统和软件的工作变得更加简单和可靠。多核CPU，更大的Cache，硬件连接间的FIFO，DMA，硬件时间戳，硬件mailbox，互斥锁，信号量等，使得多模块通信更加便捷高效，多任务共同执行的时间更有保证，系统变得更加实时，更加高效率，更加容错和即时对错误进行处理。这时候，我们把这种改进的硬件看做是承载着数据流的主通路，而CPU上运行的软件更多的是在配置，调整，和应对特殊的场景和错误。 软件更多地是考虑系统的功能需求实现，而更快更好的硬件，使得较为通用的操作系统策略和软件开发模型也可以更好地适用于工业系统。比如车载以太网凭借AVB 802.1AS（gPTP），在Linux里也可以很好地实现系统间时钟同步，这种同步的精度可以达到或好于微秒级别。

根据公开的数据，全球有超过1.5亿辆汽车搭载了QNX系统，其中包含通用，丰田，奥迪，福特等大量知名车厂。根据IHS汽车市场的调查的数据，2019年有超过4200万辆汽车的中控娱乐系统采用QNX，同一年搭载Linux的中控系统是4100万辆。预计2020年搭载Linux的中控系统将超过QNX。

ZDNet在2020年刊文说： 并不仅仅是特斯拉，事实上，举例来说，奥迪，奔驰，现代，丰田，都已经在依赖Linux。

为什么？ AGL执行董事Dan Cauchy在一份声明中说：“汽车制造商正在成为软件公司，就像在技术行业一样，他们也意识到开源是前进的道路。” 汽车公司知道，尽管销售能力强大，但客户还需要智能信息娱乐系统，自动安全驾驶功能以及最终的无人驾驶汽车。 Linux和开源公司可以为他们提供所有这些服务。

(来源：zdnet.com/article/its-a... )

Linux的开发社区非常广泛，开发人员众多，API非常丰富，除了高度兼容POSIX API便于跨平台移植，也支持Linux平台特有的一些API 以实现高性能或者特殊的操作。几乎所有成熟稳定的计算机视觉系统都方便移植给Linux，从比如OpenCV的不同版本，Caffe, Tensorflow, PyTorch等深度学习框架，从Linux开发平台迁移到Linux的嵌入式运行平台相对更为容易。从机器人框架ROS，从流媒体框架gstreamer, 包括GPU，Ethernet,WiFi, IMU等外设驱动。也包含多种面向汽车以太网等的应用协议AVB，SOME/IP。

如果看最新款的硬件设备驱动程序，或者特殊的芯片定制开发而言，对于有能力的团队，较容易通过配置Linux内核和驱动得到自己所需要的功能支持。对于比如一个最新款802.11AX Wifi芯片的支持也许几天就可以做好，因为同样的芯片加驱动可能在Android手机已经量产了。而对于所有商用RTOS而言，可能都面临着额外付费，比如需要让硬件厂商针对你的商用RTOS提供特殊的驱动程序支持。

开源库不一定能够带来100%完美的代码，但是提供了一种快速上手，快速展示概念的方式。在此基础上，中小型研发团队比如创业公司可以投入自己的研发力量，进行快速改善，从而得到自己的软件体系。

## 四．安全性和可靠性

“安全”这个词通常有两个不同的含义：信息安全（Cyber security）和功能安全（Functional Safety）。

信息安全方面有着很多基础工作，需要芯片硬件，特殊的安全或加密硬件，以及软件协议共同来保证。就操作系统本身而言，QNX，GHS等商用RTOS一直宣传自己的安全特性，系统有最少的漏洞，相比起来，开源的Linux以及基于Linux的Android，则有很多漏洞可以被黑客利用。一种理论说QNX采用微内核设计，其内核代码量非常小，而且安全权限设置合理，代码质量很高又通过很多工具检查，所以相对于Linux的安全性更好。在这里我们并不对这种说法提出质疑，因为相比起来，Linux的版本实在是太多了，你很难说出哪种Linux版本是最安全的。但对于已经公开的安全漏洞，我们很容易查到Linux特定内核版本是否存在，以及哪个版本可以解决：
linuxkernelcves.com/cve...

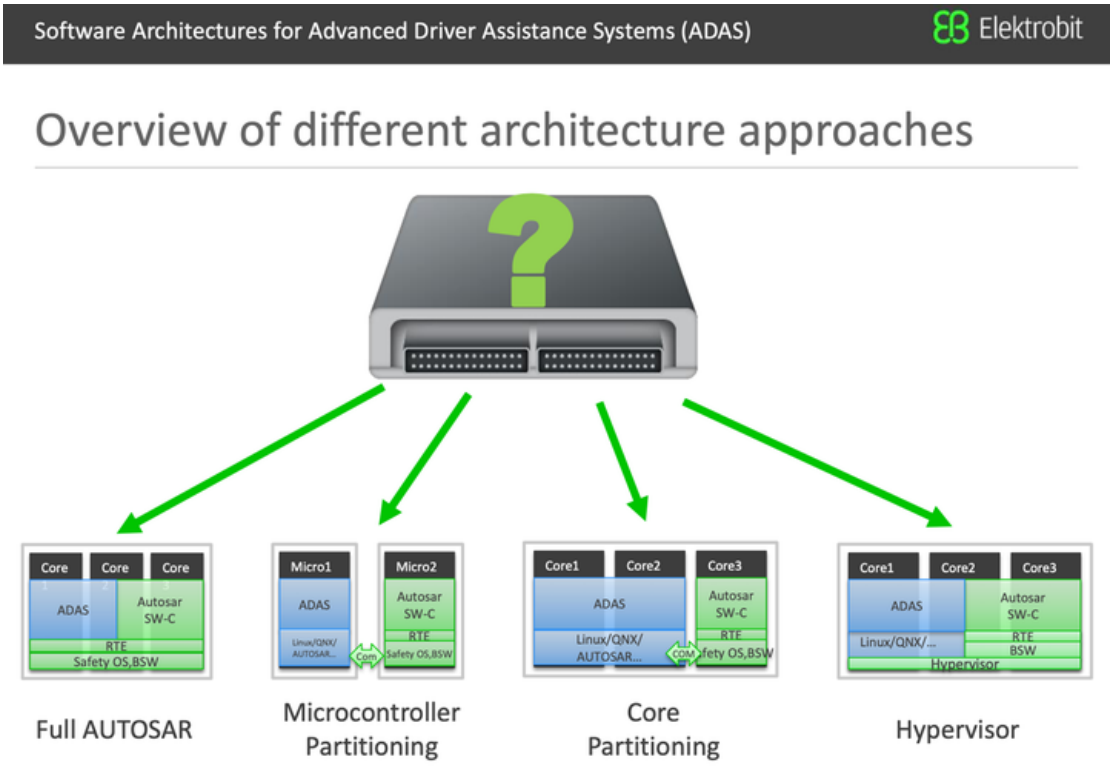
而且Linux系统下也有相应工具去执行自动扫描。这些新的安全漏洞是否影响其他的操作系统？ 不一定，但有的比如涉及到ARM架构本身的安全漏洞的补丁，在Linux内核以及工具链上很快可以得到解决，反过来看商业版本的OS什么时候可以提供补丁却是不一定。

应用广泛的OS有可能更容易被黑客当做靶子，但脱离具体的硬件来谈操作系统有多安全是没有意义的。信息安全就像“水桶理论”，系统的安全级别永远是最低的那一环节决定。实际的产品中，安全启动需要硬件和bootloader软件协同设计才可以完成，而安全环境(TEE)则需要ARM TrustZone或TPM芯片或HSM，来搭配相关的软件模块来实现。这些环节需要芯片，操作系统，设备驱动程序，专用加解密库等多个要素，最后怎样最安全，其实还是要靠清晰的设计和充分的测试。不可轻易下结论，哪个OS更为安全。

至于功能安全方面，有的RTOS比如QNX和GHS的内核可达ASIL-D级别，并有对应的认证。这方面Linux尚有差距，目前尚未获得ISO26262或者IEC61508相关的安全级别认证。但许多业界专家和研究机构，一直在孜孜不倦地努力。前面我们看到的Linux在实时性的突破也有他们的一份功劳。

Lukas Bulwahn博士，是BMW Car IT公司的功能安全软件专家，也是Linux内核实现实时性，高可靠技术的领头人。他参与了Adaptive AUTOSAR的定义与贡献，以及对OSADL组织的着眼于提升Linux实时性，可靠性的SIL2LinuxMP项目做出很多贡献，以及参与BMW公司在自动驾驶技术所需要的操作系统功能安全的研究。Bulwahn博士的原话：用于安全关键系统的Linux和主流Linux的差别就是看你如何使用它。这需要你了解你的系统，也了解Linux。

德国汽车电子软件公司Elektrobit(Continental的子公司)给出了ADAS系统的软件架构：




（来源： ADAS-architecture-ospert15-talk-keynote, Elektrobit）

从左到右列举列四种架构，经典AUTOSAR所对应的是轮转调度的简单RTOS，第三个内核划分的设计就是采用Linux或者QNX做为基础OS，运行ADAS在性能核上，而AUTOSAR运行在独立的核上。最后一个采用hypervisor的架构，目前更多地是在中控系统上见到，多个核的计算能力得到了虚拟化，在这种情况仍然是Linux或者QNX运行ADAS。

如果今天你考虑到立刻要对ADAS和自动驾驶系统的操作系统弄内核做功能安全认证ASIL-B以上，而且希望付费使用成熟稳定的商业操作系统(包括开发工具费，技术支持费用和版税),那么QNX可能是你的最佳选择。但如果你在专用的视觉感知处理器或FPGA上，实现了硬件pipeline一体化，所以Linux变成了运行在控制器上给系统做配置，调度，但Linux的实时性以及可靠性已经不对系统核心起到关键作用，那么用Linux也不会有什么技术风险。

2019年2月，Linux基金会发起了ELISA计划(Enabling Linux in Safety Applications)，研究如何把Linux用于安全关键类应用，并通过相关的功能安全认证。此项计划发起者有ARM, BMW Car IT, KUKA和丰田汽车等，随后ADIT（Bosch和Denso的合资公司）,Intel, Eletrobit, Mentor，AGL等也纷纷加入。此项目将采用基于OSADL的研究成果，继续提升安全可靠性。通过创建相关的工具，流程和文档，降低相关企业的开发成本，并加速复杂系统功能安全的实现，使得Linux更容易地被用于机器人，医疗，智能制造，交通运输，自动驾驶和航空等行业。ELISA还会促进汽车标准组织包括ISO26262考虑扩展已有的功能安全标准，以包含对开源软件的认证。这将引来汽车行业的一次重大革新，中国汽车相关企业也将有更多基于Linux的系统安全架构和流程可以复用，更好地重用成熟软件和技术，建设软件开发团队和降低开发成本。

## 五．总结

不同操作系统具有不同的特点，适合不同的系统架构，并没有哪一种是绝对好。如果我们把问题范围缩小到智能驾驶相关的舱内娱乐和自动驾驶算法来说，采用QNX是基本可行的，采用Linux并且加上相应的优化技术上也是可行的。

对于中国车厂和中国Tier1来讲，成本，开发时间，团队熟悉程度，可以凭借的软件资源等，也都非常重要。既可以通过商业授权使用成熟稳定商业操作系统QNX，也可以考虑参考特斯拉的做法，组建自己的软件队伍，深度定制Linux作为操作系统平台，也不失为一种好的选择，特别是项目没有对操作系统的功能安全等级提出认证要求。

纸上得来终觉浅，绝知此事要躬行，任何好的想法都离不开实践去摸索和检验。对于本文尚未详细讨论的问题，有关汽车智能驾驶功能安全的系统架构，我们后续会有相关文章进行探索，欢迎关注。


**关于安霸半导体：**

安霸半导体(Ambarella)的AI芯片广泛应用于高清视频类及计算机视觉感知类产品，包括视频安防、无人机、车用高级驾驶辅助系统（ADAS）、驾驶员监控系统（DMS）、智能安全座舱解决方案、电子后视镜系统（CMS）、自动驾驶和机器人等。安霸半导体的SoC可用于低功耗设计、超高清视频压缩、高质量图像处理及带人工智能加速器的车用产品，通过高性能计算给汽车智能视觉感知方案赋能。更多信息请访问 https://www.ambarella.com

发布于 2020-08-31 19:43

自动驾驶 AI芯片 操作系统

写下你的评论...

还没有评论，发表第一个评论吧

## 推荐阅读


**纯干货，汽车芯片究竟缺的是什么？自动驾驶域控制器才是关键**

汽车人参考　　发表于汽车人参考

**汽车芯片：自动驾驶浪潮之巅**

王博Kin...　　发表于无人驾驶干...


**深度解析|自动驾驶技术——芯片的过去、现在和未来**

Ares ...　　发表于汽车自