道哥的第 029 篇原创

一、前言

前几天和一个在某研究所的发小聊天，他说：现在的航空、航天和导弹等武器装备中，控制系统几乎都是用单片机，而不是嵌入式系统。

乍一听，和我们的直觉有矛盾啊：那么高大上的设备，其中的控制逻辑一定很复杂，不用嵌入式系统怎么来完成那么复杂的功能控制啊？然后仔细了解了一下，才明白答案是：安全+可控。

这篇文章我们就来聊一下关于单片机与嵌入式、操作系统与 RTOS 之间的那些事！通过这篇文章，让你操作系统的实时性有一个系统、全面的理解！

二、关于单片机与嵌入式系统之间界定

说实话，关于它俩的区分，没有人可以给出一个标准的、正确的答案。每个人理解的单片机与嵌入式系统，都是略有差别的。

抛开硬件，从应用程序开发的角度来看，我是这样来理解的：

单片机：可以直接使用状态机来实现程序框架，也可以利用一些 RTOS(ucOS、FreeRTOS、vxWorks、RT-Thread)等来完成一些调度功能。

嵌入式系统：利用嵌入式 Linux 操作系统以及一些变种来编写应用程序。

我知道自己的理解可能是不对的，至少不严谨、范围狭隘，既然没有标准答案，那姑且引用维基百科中的定义吧，毕竟概念是死的，更重要的是我们如何根据实际的需要来进行选择。

1. 单片机

> 1. 单片机，全称单片微型计算机（single-chip microcomputer），又称微控制器单元 MCU（microcontroller unit）。
> 2. 把中央处理器、存储器、定时/计数器、各种输入输出接口等都集成在一块集成电路芯片上的微型计算机。
> 3. 由于其发展非常迅速，旧的单片机的定义已不能满足，所以在很多应用场合被称为范围更广的微控制器；

2. 嵌入式系统

> 1. 嵌入式系统（Embedded System），是一种嵌入机械或电气系统内部、具有专一功能和实时计算性能的计算机系统。
> 2. 嵌入式系统常被用于高效控制许多常见设备，被嵌入的系统通常是包含数字硬件和机械部件的完整设备，例如汽车的防锁死刹车系统。
> 3. 现代嵌入式系统通常是基于微控制器（如含集成内存和/或外设接口的中央处理单元）的，但在较复杂的系统中普通微处理器（使用外部存储芯片和外设接口电路）也很常见。

3. 嵌入式Linux

> 1. 嵌入式Linux（英语：Embedded Linux）是一类嵌入式操作系统的概称，这类型的操作系统皆以Linux内核为基础，被设计来使用于嵌入式设备。
> 2. 与电脑端运行的linux系统本质上是一样的，虽然经过了一些功能上的裁剪，但是本质上是一样的，主要利用 Linux 内核中的的任务调度、内存管理、硬件抽象等功能。

4. RTOS

> 1. 实时操作系统（RTOS），又称即时操作系统，它会按照排序运行、管理系统资源，并为开发应用程序提供一致的基础。
> 2. 实时操作系统与一般的操作系统相比，最大的特色就是“实时性”，如果有一个任务需要执行，实时操作系统会马上（在较短时间内）执行该任务，不会有较长的延时。这种特性保证了各个任务的及时执行。

三、非实时、软实时、硬实时

首先要明白什么叫实时性？实时性考虑的不是速度、性能、吞吐量，而是确定性，也就是说：当一个事件发生的时候，可以确定性的保证在多长时间内得到处理，只要能满足这个要求，就可以成为硬实时。比如：

> 操作系统1：当中断发生时，可以保证在 1 秒内得到这里，那么它就是硬实时系统，虽然响应时间长，但它是确定的；
> 操作系统2：当中断发生时，几乎都可以在 1 毫秒内完成，那么那就不能成为硬实系统，虽然响应时间短，但是它不确定。

也看到有文章说：应该取消软实时这个模棱两可的说法，要么是实时，要么是非实时！

操作系统包含的功能很多：任务调度、内存管理、文件管理等等，其中最核心的就是任务调度，这也是非实时、软实时、硬实时的最大区别。

也就是说，衡量实时性的指标就是：

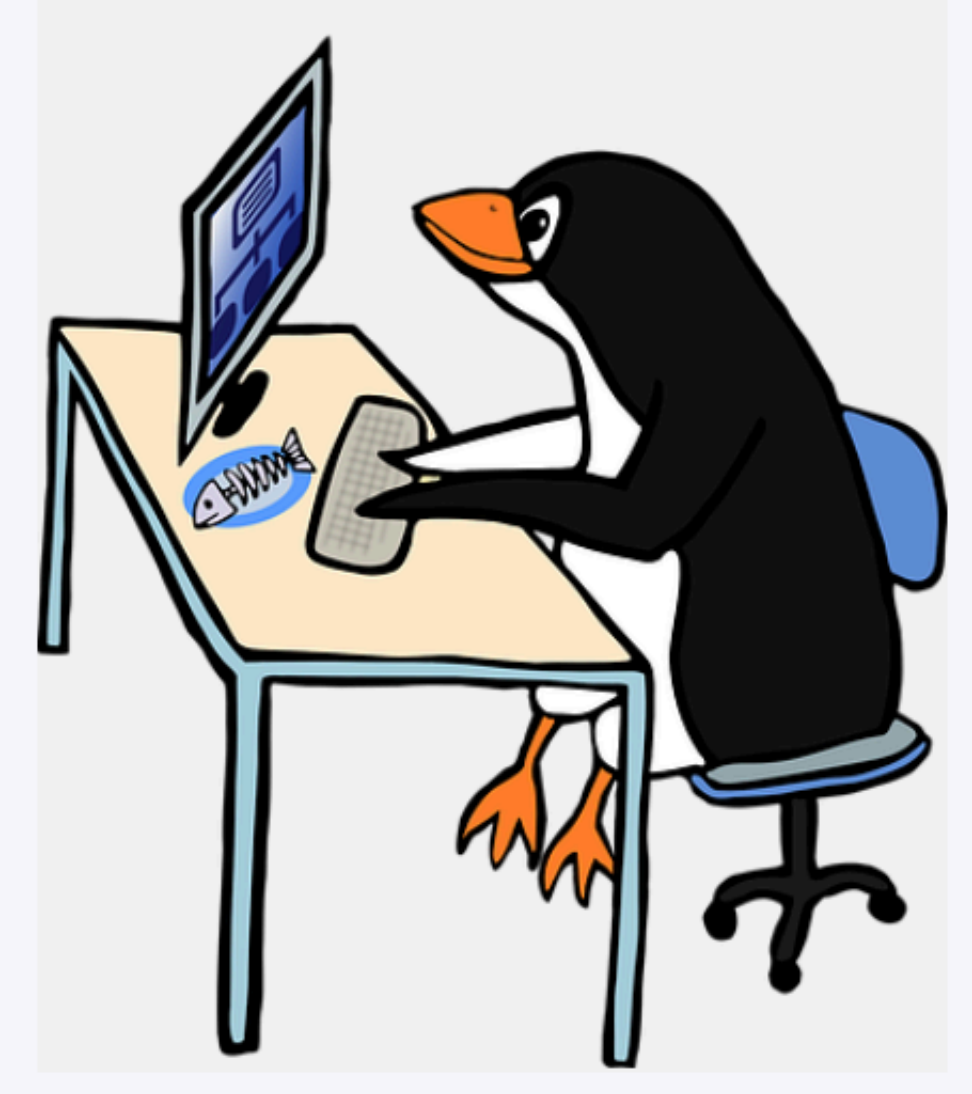
1. 中断延时：一个外部事件引发的中断发生时，到相应的中断处理程序第一条指令被执行时，所经过的时间；
2. 任务抢占延时：当一个高优先级的任务准备就绪时，从正在执行的低优先级任务中抢夺 CPU 资源所经过的时间；

不同的操作系统，其任务调度机制也是不一样的，而这个调度机制的策略，又是与实际的使用场景相关的。因此，并不存在哪个好、哪个不好这样的说法，合适的就是最好的！

比如：我们的桌面系统，需要考虑的是多任务、并发，需要同时执行多个程序，哪个程序慢一点，用户无所谓，甚至觉察不到；但是对于一个导弹控制系统，当一个外部传感器输入信号，触发一个事件时，对应的处理必须立刻执行，否则耽搁 1 毫秒，结果可能就是差之千里！

四、x86 Linux 系统的调度策略

我们日常使用的 PC 机，它的主要目标是并行执行多任务，强调的是吞吐率(尽可能多的执行很多应用程序的代码)，因此，采用的是分时操作系统，也就是每个任务都有一个时间片，当一个任务分配的时间片用完了，就自动换出（调度），然后执行下一个任务。



我们平常在写 x86 平台上写普通的客户端程序时，很少需要指定应用程序的调度策略和优先级，使用的是系统默认的调度机制。反过来说，也就是在某些需要的场合下，是可以设置进程的调度策略和优先级的。

例如在 Linux 系统中，可以通过 `sched_setscheduler()` 系统函数 设置 3 种调度策略：

> 1. SCHED_OTHER: 系统默认的调度策略，计算动态优先级（counter+20-nice），当时间片用完之后放在就绪队列尾；
> 2. SCHED_FIFO: 实时调度策略，根据优先级进行调度，一旦占用CPU就一直执行，直到自己放弃执行或者有更高优先级的任务需要执行；
> 3. SCHED_RR: 也是实时调度策略，在 SCHED_FIFO 的基础上添加了时间片。在执行时，可以被更高优先级的任务打断，如果没有更高优先级的任务，那么当任务的执行时间片用完之后，就会查找相同优先级的任务来执行。

1. 为什么 Linux 系统是软实时的?

可能有小伙伴会有疑问：既然 Linux 系统中提供了 SCHED_FIFO 基于优先级的调度策略，为什么仍然不能称之为真正的硬实时操作系统？这就要从 Linux 的发展历史说起了。

Linux 操作系统在设计之初，就是为了桌面应用而开发的，在那个时代，多个终端(电传打字机和屏幕)连接到同一个电脑主机，需要处理的是多任务、并行操作，并不需要考虑实时性，因此，在 Linux 内核中的一些基因，严重影响了它的实时性，例如有如下几个因素：

(1) 内核不可抢占

我们知道，一个应用程序在执行时，可以在用户态和内核态执行(当调用一个系统函数，例如：write 时，就会进入内核态执行)，此时任务是不可抢占的。

即使有优先级更高的任务准备就绪，当前的任务也不能立刻停止执行。而是必须等到当前这个任务返回到用户态，或者在内核态中需要等待某个资源而睡眠时，高优先级任务才可以执行。

因此，这就很显然无法保证高优先级任务的实时性了。

(2) 自旋锁

自旋锁是用于多线程同步的一种锁，用来对共享资源的一种同步机制，线程反复检查锁变量是否可用。由于线程在这一过程中保持执行，因此是一种忙等待。一旦获取了自旋锁，线程会一直保持该锁，直至显式释放自旋锁。

自旋锁避免了进程上下文的调度开销，因此对于线程只会阻塞很短时间的场合是有效的，也就是说，只能在阻塞很短的时间才适合使用自旋锁。

但是，在自旋锁期间，任务抢占将会失效，这就是说，即使自旋锁的阻塞时间很短，但是这仍然会增加任务抢占延时，让调度变得不确定。

(3) 中断的优先级是最高的

任何时刻，只要中断发生，就会立刻执行中断服务程序，也就是中断的优先级是最高的。只有当所有的外部中断和软终端都处理结束了，正常的任务才能得到执行。

这看起来是好事情，但是想一想，如果有比中断优先级更高的任务呢？假如系统在运行中，网口持续接收到数据，那么中断就一直被执行，那么其他任务就可能一直得不到执行的机会，这是影响 Linux 系统实时性的巨大挑战。

(4) 同步操作时关闭中断

如果去看 Linux 内核的代码，可以看到在很多地方都执行了关中断指令，如果在这期间发生了中断，那么中断响应时间就没法保证了。

2. Linux 系统如何改成硬实时？

以上描述的几个因素，对 Linux 实现真正的实时性构成了很大的障碍，但是现实世界又的确有很多场合需要 Linux 具有硬实时，那么就要针对上面的每一个因素提出解决方案。

目前主流的解决方案有 2 个：

> 1. 单内核解决方案：给 Linux 内核打补丁，解决上面提到的几个问题，例如：RT-Preempt;
> 2. 双内核解决方案：在硬件抽象层之上，运行 2 个内核：实时内核 + Linux 内核，它们分别向上层提供 API 函数，例如：Xenomai;

这 2 种解决方案分别有不同的实现，从调研情况来看，RT-Preempt 和 Xenomai 是使用比较多的，下面分别来看一下他们的优缺点。

（1）RT-Preempt

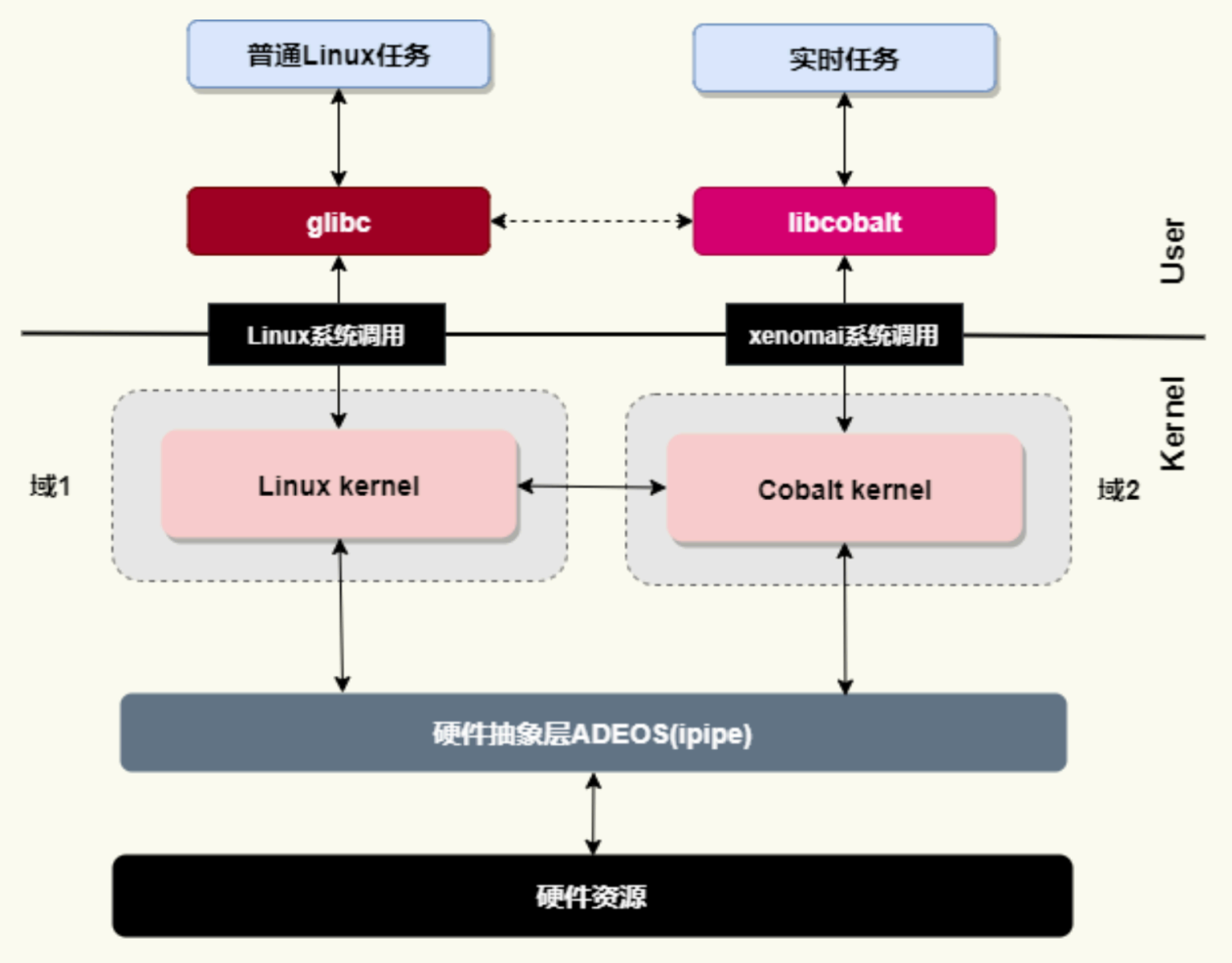
这种方式主要是对 Linux 内核进行打补丁，解决了上面所说的几个问题：内核不可抢占、自旋锁、关中断以及终端优先级的问题。

至于每一个问题是如何解决的，由于篇幅关系，这里就不介绍了，感兴趣的小伙伴如果需要的话，可以深入了解一下。

由于是直接在 Linux 内核上打补丁(以后肯定会合并到主分支中的)，因此对于应用程序开发来说，操作系统向上层提供的 API 接口函数可以保持不变，这对应用程序开发来说是一件好事情。

（2）Xenomai

Xenomai是一个 Linux 内核的实时开发框架，它希望通过无缝地集成到 Linux 环境中来给用户空间应用程序提供全面的，与接口无关的硬实时性能。下面是 Xenomai 的架构图：



在硬件抽象层之上，是 2 个并列的域(内核)，这 2 个内核分别向上层提供自己的 API 接口函数。

图中 glibc 是 Linux 系统提供的库函数，应用程序通过调用库函数和系统调用来编写程序。

Xenomai 也提供了相应的库函数 libcobalt ，这个库函数是需要我们在用户层编译、安装的，就像安装第三方库一样。

此外，Xenomai 还参考不同的操作系统风格，提供了好几套 API 函数(之前的说法是：皮肤)，API 接口函数在这里：

Xenomai 3.0.5

从图中可以看到，Alchemy API 这套接口提供的功能更完善，提供了：定时器、内存管理、条件变量、事件、互斥锁、消息队列、任务(可以理解为线程)等 API 函数。
这一套 API 函数中具体的功能与 POSIX 标准大体相同，在一些细节上存在一些差异。

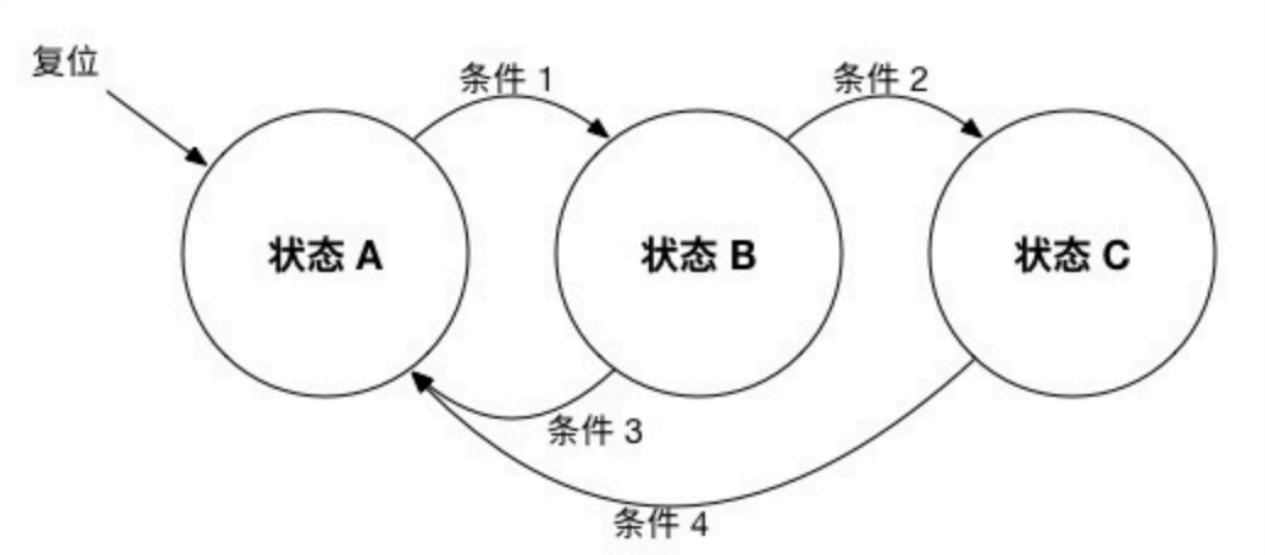
由于 Xenomai 向应用层提供的 API 函数是独立的一套，因此，如果我们需要创建实时任务，那么就要调用这一套接口函数来创建任务，包括使用其中的一些资源(例如：内存分配)。而且文档中也提出了一些注意点，例如：某些资源不能在 Xenomai 与 Linux 系统之间混用。

五、RTOS 的优势

上面已经说到，Linux 桌面系统的主要目标是吞吐量，在单位时间内执行更多的代码。

但是对于单片机来说，首要目标不是吞吐量，而是确定性，因此衡量一个实时操作系统坚固性的重要指标，是系统从接收一个任务，到完成该任务所需的时间。也就是说，任务调度才是第一考量要素。

在单片机开发中，一般有 2 种编程模型：基于状态机(裸跑)，基于 RTOS。



如果基于状态机，就不存在任务调度问题了，因为只有一个执行序列，所有的操作都是串行执行的，唯一需要注意的控制流程就是中断处理。

如果基于 RTOS，主要利用的就是任务调度，实现真正的硬实时。这方面最牛逼的就是VxWorks了，当然价格也是非常可观的，有些公司购买之后，甚至会把除了任务调度模块之外的其他模块全部重写一遍，这也足以证明了 VxWorks 在任务调度处理上的确很厉害，这也是它的看家本领!

当然，对于简单、需要严格控制执行序列的关键程序来说，使用有限状态机的编程框架，一切都在自己的掌握中。只要代码中没有 bug，那么理论上，一切行为都是在控制之中的，这也是为什么很多军事设备上使用单片机的原因！

六、总结

关于任务调度的问题，是一个操作系统的重中之重，其中需要学习的内容还有很多，最近刚买了一本陈海波老师的新书，也就是华为的鸿蒙系统背后的灵魂人物。

如果有新的学习心得，再跟大家分享。

参考文献：

> https://linuxfoundation.org/blog/intro-to-real-time-linux-for-embedded-developers/
> https://wiki.archlinux.org/index.php/Realtime_kernel_patchset
> http://www.faqs.org/faqs/realtime-computing/faq/
> https://xenomai.org/documentation/xenomai-3/html/README.INSTALL/

好文章，要转发；越分享，越幸运！

推荐阅读

【C 语言】

C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻
原来gdb的底层调试原理这么简单
一步步分析-如何用C实现面向对象编程
提高代码逼格的利器：宏定义-从入门到放弃

利用C语言中的setjmp和longjmp，来实现异常捕获和协程

【应用程序设计】

都说软件架构要分层、分模块，具体应该怎么做(一)
都说软件架构要分层、分模块，具体应该怎么做(二)
物联网网关开发：基于MQTT消息总线的设计过程(上)
物联网网关开发：基于MQTT消息总线的设计过程(下)
我最喜欢的进程之间通信方式-消息总线

【物联网】

关于加密、证书的那些事
深入LUA脚本语言，让你彻底明白调试原理

【胡说八道】

以我失败的职业经历：给初入职场的技术人员几个小建议