Linux TTY framework(5)_System console driver

作者：wowo 发布于：2016-10-29 22:43 分类：TTY子系统

1. 前言

由[1]中的介绍可知，Linux kernel的console框架，主要提供“控制台终端”的功能，用于：

> 1）kernel日志信息（printk）的输出。
>
> 2）实现基础的、基于控制台的人机交互。

本文将从console driver开发者的视角，介绍：console有关的机制；编写一个console驱动需要哪些步骤；从用户的角度怎么使用；等等。

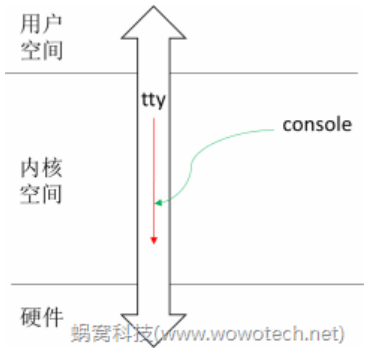
2. 设计思路介绍

不知道大家是否有这样的疑问：既然已经有了TTY框架，为什么要多出来一个console框架，为什么不能直接使用TTY driver的接口实现console功能？

确实，由[2]中的介绍可知，TTY框架的核心功能，就是管理TTY设备，并提供访问TTY设备的API（如数据收发）。而console的两个功能需求，“日志输出”就是向TTY设备发送数据，“控制台人机交互”就是标准的TTY功能。因此从功能上看，完全可以直接使用TTY框架的API啊。

不过，既然存在，一定有其意义。内核之所以要抽象出console框架，思路如下：

1）Linux kernel有一个很强烈的隐性规则----内核空间的代码不应该直接利用用户空间接口访问某些资源，例如kernel代码不应该直接使用文件系统接口访问文件（虽然它可以）。回到本文的场景里面，TTY框架通过字符设备（也即文件系统）向用户空间提供接口，那么kernel的代码（如printk），就不能直接使用TTY的接口访问TTY设备，怎么办呢？开一个口子，从kernel里面再拉出一套接口，这就是console框架，如下图所示：



2）console框架构建在TTY框架之上，大部分的实现（特别是访问硬件的部分）都和TTY框架复用。

3）系统中可以有多个TTY设备，只有那些附加了console驱动的设备，才有机会成为kernel日志输出的目的地，有机会成为控制台终端。因此，console框架变相的成为管理TTY设备的一个框架。

4）驱动工程师在为某个TTY设备编写TTY driver的时候，会根据实际的需求，评估该TTY设备是否可能成为控制台设备，如果可能，则同时为其编写system console driver，使其成为候选的控制台设备。系统工程师在系统启动的时候，可以通过kernel命令行参数，决定printk会在哪些候选设备上输出，那个候选设备最终会成为控制台设备。示意图如下：

站内搜索

请输入关键词 搜索

功能

留言板
评论列表

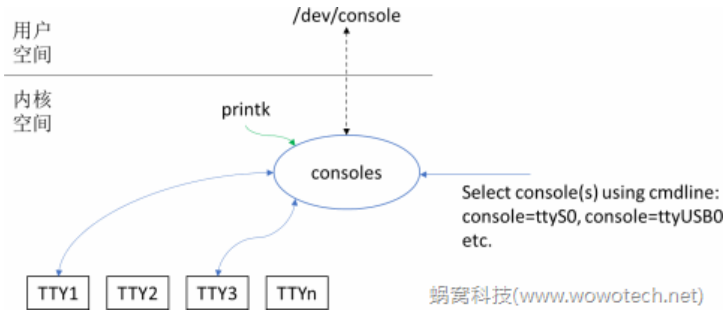
最新评论

luke
因公司网络限制，文章里的图片不能显示，看的好辛苦啊！！
linuxer
@hit201j：已经修改，多谢！
bigpillow
Hi Linuxer, 我们这边有写一套完整的GPIO &...
hit201j
错别字：“还以一个就是bootloader传递过来的bl...
GrayMonkey
膜拜大佬，早找到你的文章就好了,Android开发一枚，一直...
fy
@wowo：是的。我只是就这里的“禁止抢占”非必要，问问wo...

文章分类

Linux内核分析(9)
统一设备模型(13)
电源管理子系统(42)
中断子系统(14)
进程管理(13)
内核同步机制(17)
GPIO子系统(5)
时间子系统(14)
通信类协议(7)
内存管理(20)
图形子系统(1)
文件系统(3)
TTY子系统(5)
u-boot分析(3)
Linux应用技巧(13)
软件开发(6)
基础技术(13)
蓝牙(16)
ARMv8A Arch(13)
显示(3)
基础学科(9)
技术漫谈(12)
项目专区(0)
X Project(28)

随机文章



3. 核心数据结构

理解了console框架的设计思路之后，再来看它的实现，就很简单了。其核心数据结构为struct console，如下：

```
/* include/linux/console.h */

struct console {
        char    name[16];
        void    (*write)(struct console *, const char *, unsigned);
        int     (*read)(struct console *, char *, unsigned);
        struct tty_driver *(*device)(struct console *, int *);
        void    (*unblank)(void);
        int     (*setup)(struct console *, char *);
        int     (*match)(struct console *, char *name, int idx, char *options);
        short   flags;
        short   index;
        int     cflag;
        void    *data;
        struct   console *next;
};
```

该数据结构中经常被使用的字段有：

name，该console的名称，配合index字段，可用来和命令行中的“console=xxx”中的“xxx”匹配，例如：

> 如果name为“ttyXS”，index为大于等于0的数字（例如2），则可以和“console=ttyXS2”匹配；
>
> 如果name为“ttyXS”，index为小于0的数字（例如-1），则可以和“console=ttySXn“（n=0,1,2…）任意一个匹配。

write，如果某个console被选中作为printk的输出，则kernel printk模块会调用write回调函数，将日志信息输出到。

device，获取该console对应的TTY driver，用于将console和对应的TTY设备绑定，这样控制台终端就可以和console共用同一个TTY设备了。

setup，用于初始化console的回调函数，console driver可以在该回调函数中对硬件做出现动作。可以不实现，如果实现，则必须返回0，否则该console不可用。

flags，指示属性的flags，常用的包括：

> CON_BOOT，该console是一个临时console，只在启动的时候使用，kernel会在真正的console注册后，把它注销掉。
>
> CON_CONSDEV，表示该console会被用作控制台终端（和/dev/console对应），对应命令行中的最后一个，例如“console=ttyXS0 console=ttyUSB2”中的ttyUSB2。
>
> CON_PRINTBUFFER，如果设置了该flag，kernel在该console被注册的时候，会将那些被缓存到buffer中的之前的日志，统统输出到该console上。通常注册的console，如串口console，都会设置该flag，以便可以看到console注册前的日志输出。
>
> CON_ENABLED，表示该console正在被使用。

X-011-UBOOT-使用bootm命令启动kernel(Bubblegum-96平台)
Process Creation（一）
为什么会有文件系统(二)
linux kernel的中断子系统之（九）：tasklet
中断上下文中调度会怎样？

文章存档
2017年11月(1)
2017年10月(1)
2017年9月(5)
2017年8月(4)
2017年7月(4)
2017年6月(3)
2017年5月(3)
2017年4月(1)
2017年3月(8)
2017年2月(6)
2017年1月(5)
2016年12月(6)
2016年11月(11)
2016年10月(9)
2016年9月(6)
2016年8月(9)
2016年7月(5)
2016年6月(8)
2016年5月(8)
2016年4月(7)
2016年3月(5)
2016年2月(5)
2016年1月(6)
2015年12月(6)
2015年11月(9)
2015年10月(9)
2015年9月(4)
2015年8月(3)
2015年7月(7)
2015年6月(3)
2015年5月(6)
2015年4月(9)
2015年3月(9)
2015年2月(6)
2015年1月(6)
2014年12月(17)
2014年11月(8)
2014年10月(9)
2014年9月(7)
2014年8月(12)
2014年7月(6)
2014年6月(6)
2014年5月(9)
2014年4月(9)
2014年3月(7)
2014年2月(3)
2014年1月(4)

4. 接口说明

4.1 console driver的注册接口

对具体的console driver来说，只需要关心console的注册/注销接口即可：

```
/* include/linux/console.h */

extern void register_console(struct console *);
extern int unregister_console(struct console *);
```

在正确填充struct console变量之后，通过register_console接口将其注册到kernel中即可。该接口将会完成如下的事情：

> 检查该console的name和index，确认之前没有注册过，否则注册失败；
>
> 如果该console为boot console（CON_BOOT），确认是第一个注册的boot console，否则注册失败；
>
> 如果系统中从来没有注册过console，则将第一个被注册的、可以setup成功（.setup为NULL或者返回0）的console作为正在使用的console，并使能之（CON_ENABLED）；
>
> 和command line中的“console=xxx”最对比，使能那些在命令行中指定的console；
>
> 查找在command line中指定的最后一个console，并置位其CON_CONSDEV flag，表明选择它为控制台console。

4.2 用户层面接口

系统console的使用控制，主要由命令行参数（一般都是bootloader传递而来的）指定，总结如下：

1）如果某个console只需要在启动的时候使用，则需要在注册console的时候，置位其CON_BOOT标志，例如[3]中介绍的early console。

2）如果系统中注册了多个console，可以通过命令行参数指定使用哪个或者哪些，kernel的日志将会输出到所有在命令行指定了的console上面。

3）命令行中指定的最后一个console，将会作为控制台console，应用程序打开/dev/console将会打开该console对应的TTY设备（由.device回调返回的tty driver指定）。

5. console driver的编写步骤

理解了system console的原理之后，编写一个console driver就比较简单了，包括：

1）如果希望该console可以作为系统控制台（/dev/console），则必须先实现该console对应的TTY设备的TTY driver。

2）定义一个console变量，并根据实际情况填充对应的字段，包括name、index、setup（可选）、write、device（可选）等。

3）调用register_console将其注册到kernel中即可。

6. 参考文档

[1] Linux TTY framework(3)_从应用的角度看TTY设备

[2] Linux TTY framework(4)_TTY driver

[3] X-012-KERNEL-serial early console的移植

标签: Linux Kernel 内核 driver console

« X-014-KERNEL-ARM GIC driver的移植 | 进程管理和终端驱动：基本概念»

评论：

lucky
2017-11-08 17:42

Hi wowo,
你上面有提到:Linux kernel在启动时会打开"/dev/console"，然后dup出来3个文件句柄（我们熟悉的0，1，2）交给init进程使用，然后一直被继承（除非被应用程序显式的修改）；
有个问题想请教下, 就console的打印log功能来讲, 内核空间的log printk和用户空间的log都是输出到 console上吗?也就是都会最终调用到 struct console 的write函数? 最终log是输出到哪里呢? 内存里还是某块存储空间? 这个log和平时android平台常抓的串口log(串口log是最终写到物理串口设备上)是一回事吗?

回复

wowo
2017-11-09 18:23

@lucky：用户空间的日志输出，是通过VFS接口，经由TTY driver，给到具体的console driver。和kernel printk（ struct console 的write函数）的路径不一样。
至于log输出到哪里，由相应的driver决定。
至于android的logcat，不是一个层次的事情，不要搅在一起理解……

回复

xmj
2016-12-29 15:56

hi，wowo
看了你这篇文字有疑问，能否解答一下呢
1. console的write和它里面的tty_driver write有什么区别吗
2. 般情况下我们看串口log，手机bootloader起来就可以看了，这个时候console没初始化完成，是根据什么来输出log的吗，还有JIGTAG也可以在很早输出log？ 能说下这几个的差别吗

回复

wowo
2016-12-29 21:36

@xmj：1. 我觉得这两个没有本质的区别，是向同一个通道里写数据的两个路径，参考本文的图片1.
2. 手机bootloader一般会自己初始化串口屏输出信息；JTAG的log，我也不是特别了解，估计类似一个虚拟串口。

回复

electrlife
2016-10-31 13:58

另外对于这段时间的学习，我也总结下我对相关的几个概念的理解（请指教理解的是否正确）：

tty
从时间轴上看tty仅是电传打印机，由于广泛的作为终端设备来使用，因此成了终端的代名词。请先忘记当前你手中使用的PC机及当前使用的终端，因为我们现在所讲的是终端还是远在时代无图形界面的计算机时代。

Pty 伪终端
构造出的一对通信链路，这个链路服务于本机上的应用程序。如telnet/ssh远程登录程序，大概的过程可以描述如下：
sshd 等待client的socket connect request，如果有client申请登录，则sshd daemon会通过系统调用创建两个伪终端，然后通过getty来对相应的终端进行设置。
sshd只需要从其中特定的一个终端读出或写入数据并通过socket转发给client。此时sshd只负责数据转发，相应的prompt和登录相应的还可以由标准的getty和login来完成。
因getty login默认使用tty作为它们的输入输出，因此我们可以通过伪造一对假的终端来供它们使用。

仿真终端
随着时间的流逝，计算机发展出图形界面，于是仿真的终端应运而生，如大名鼎鼎的xterm。 仿真终端运行在用户空间的一个程序，它使用软件模拟的方式来仿真一个真的终端所具有的所有功能。比如响应调节相应的显示size等。 对于终端另一端的应用程序来说，它们甚至无法识别对面是一个假的终端应用程序。xterm捕捉用户的输入并发送到终端另一边的应用，收集终端另一边的应用的数据，并通过图形接口显示到我们LCD上。因此对于仿真终端来说也使用伪终端来构建这样的数据通路。

另外对于vt这块，暂时还不能和tty及console在源码上联系起来。

回复

wowo
2016-10-31 14:21

@electrlife：总结的很好啊，不过我觉得:从TTY技术本身看，应该没有仿真终端的概念（你这里提到的仿真终端，本质上还是一个伪终端，因为shell等传统应用一定需要一个tty设备，你不能不给它）。

回复

electrlife
2016-10-31 20:16

@wowo：对的！

对于vt是否会有相应的专题？感觉从代码上现在完全没有头绪！

回复

electrlife
2016-10-31 13:54

Hi wowo, 这段时间一直跟着你们的文章还有自己的理解，对于console发表下自己的看法，（PS 不一定正确）

首先对于console，我的理解就现在的Linux而言似乎已经退化到打印Log的用途，简而理解仅仅作为printk输出的对象。如果这个理解正确，那么对console我个人觉得当前似乎和tty没有什么关系，从printk的代码我仅仅看出console的write等函数被调用，完全不见terminal的足迹。
如果非要说console和终端的联系，我觉得仅仅是和uart有联系。和tty没有什么关联。当然在处理tty时可能需要考虑下如果该tty如果作为console需要额外的特殊处理，仅此而矣！
因此，个人觉得在当下的Linux系统，console仅仅抽象了一个log输出的底层接口而矣！

回复

wowo
2016-10-31 14:19

@electrlife：其实我对TTY的熟悉程度也不高，我们刚好可以一起讨论:-)
console一直是两个功能，一个是我们熟悉的printk，这里就不提了。
另一个，console真的代表了正宗的“控制台终端”。要理解这个概念，需要围绕“/dev/console”和“console->device”来说：
“/dev/console”代表了Linux/Unix系统中正宗的“控制台终端”，不需要知道为什么，就当是一种定义；
“/dev/console”是由tty core（tty_io.c）在初始化时创建的，是一个虚拟的设备；
Linux kernel在启动时会打开“/dev/console”，然后dup出来3个文件句柄（我们熟悉的0，1，2）交给init进程使用，然后一直被继承（除非被应用程序显式的修改）；
既然“/dev/console”是一个虚拟的设备，它对应的物理设备是什么呢？你应该猜到了，根据命令行参数，调用“console->device”获取对应的struct tty_driver之后，就得到了实际的物理设备。
最后总结，如果你不想kernel在open(“/dev/console”)报错，就应该实现当前所使用的console的“console->device”回调。当然，就算打不开，也没关系，后果顶多是有些应用无法输入输出（例如init进程）。

回复

electrlife
2016-10-31 19:31

@wowo：1、我试着从你的这个角度理解了下，感觉有道理！我也试着简单跟了下open（/dev/console）代码，发现对于找到tty_driver这个数据结构时会区分console是tty device。但是对于console_device这个函数，有点疑惑？这个函数会遍历所有注册的console driver只要driver存在便返回。我的理解是console应该只会有一个，这里为什么会使用遍历！
2、对于启动参数我们传入的console=xxx, console=yyy, ...这种形式，系统会打印logs到所有我们注册的console，但系统真正的console只会是最后一个。是不是意味着前面的几个console其实是不需要tty_driver这样的数据结构的？
3、我的理解 Open(/dev/console)其实打开是对应应用程序的controlling terminal？

回复

wowo
2016-11-01 08:57

@electrlife：问题1：关于console_device，你可以去printk.c中检查一下它遍历的那个链表（console_drivers），其实是排序过的，最先拿到的，就是那些preferred driver（带CON_CONSDEV标记的）。话说这一段代码逻辑也是很巧妙的，值得学习一下。
问题2：是的。但是本着console和tty分离的原则，我们注册console的时候，其实是不假设它是否会作为控制台的，所以如果能提供，最好都提供，如果觉得没必要，不提供也ok。
问题3：(/dev/console)不是controlling terminal，（/dev/tty）才是。说实话这几个术语的中文翻译有点绕，我倒觉得，(/dev/console)翻译成工作台反而更容易理解，它的操作者是人。（/dev/tty）是控制终端，它的操作者是软件（shell）。

回复

electrlife
2016-11-01 10:26

@wowo：抱歉！能提示下排序的关键点在哪些代码上吗？真的没有看出来！

回复

wowo
2016-11-01 11:29

@electrlife：这一段代码逻辑我还没有细看，还是你看看之后给我讲解一下吧^V^

回复

发表评论：

昵称

邮件地址 (选填)

个人主页 (选填)

发表评论