[linux下热插拔事件的产生是怎样通知到用户空间，kobject\_uevent\_env之uevent\_helper](http://blog.csdn.net/bingqingsuimeng/article/details/7924300)

标签： [linux](http://www.csdn.net/tag/linux)[struct](http://www.csdn.net/tag/struct)[filter](http://www.csdn.net/tag/filter)[path](http://www.csdn.net/tag/path)[脚本](http://www.csdn.net/tag/%e8%84%9a%e6%9c%ac)[action](http://www.csdn.net/tag/action)

2012-08-30 11:02 5831人阅读 [评论](http://blog.csdn.net/bingqingsuimeng/article/details/7924300#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/bingqingsuimeng/article/details/7924300#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

linux驱动程序之-热插拔（6） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

本人CU博客中的文章，本来是希望CU给推荐到首页的，结果周五不知道CU博客出了啥问题，居然没什么更新。所以我干脆把再把它放到这里，因为话题涉及设备驱动模块自动加载，本版已经有同学问过这方面的问题。  
  
热插拔(hotplug，打这个词的时候我常常想到热干面)不一定非要指类似U盘那样的插入拔出，此处的热插拔广义上讲，是指一个设备加入系统，内核如何通知用户空间。举个简单的例子，如果你的电脑中有块PCI网卡，针对该网卡的驱动程序以内核模块的形式被编译(obj-m)，那么Linux系统在启动过程中是如何自动加载该网卡的驱动模块呢？大家都知道现在udev负责干这事，其实除了udev，还可以有其他的手法，你自己就可以这样做。  
  
我们先讨论udev，udev最关键的东西是当系统发现一个设备时，它要能够被通知该事件，一旦它知道了这件事，那么余下的事情就都好说了，无非是个如何查找模块并加载的过程。所以我们看到，这里的关键是热插拔事件的通知机制。Linux的设备模型为此提供了非常完美的支持，其原理其实发源于kset这一层，对此在《深入Linux设备驱动程序内核机制》一书中有详细的描述，虽然这部分看起来蛮复杂，貌似挺能吓唬住一些新手，其实说白了，要点就是通过sysfs建立关系，沟通内核与用户空间，然后就是uevent，也就是下面要说的热插拔事件。  
  
当然设备驱动程序一般不会和这些太底层的kobject/kset家伙打交道，因为更高层次的device,bus和driver把kobject/kset那一层的细节实现都给封装了起来。所以设备热插拔的uevent事件最终的源头来自于device\_add，本帖这里肯定不会讨论device与driver如何绑定那一摊子事情。下面看看device\_add的源码，是如何实现uevent机制的：

1. <drivers/base/core.c>
2. int device\_add(struct device \*dev)
3. {
4. ...
5. kobject\_uevent(&dev->kobj, KOBJ\_ADD);
6. ...
7. }

复制代码

热插拔的核心实现就那一个函数调用，这里device\_add对应的是KOBJ\_ADD，那么移除设备自然对应KOBJ\_REMOVE了。kobject\_uevent函数最终调用的是kobject\_uevent\_env，后者才是真正干事的伙计。  
下面给出kobject\_uevent\_env函数的核心框架：

1. int kobject\_uevent\_env(struct kobject \*kobj, enum kobject\_action action,
2. char \*envp\_ext[])
3. {
4. ...
5. #if defined(CONFIG\_NET)
6. /\* send netlink message \*/
7. ...
8. #endif
9. /\* call uevent\_helper, usually only enabled during early boot \*/
10. if (uevent\_helper[0] && !kobj\_usermode\_filter(kobj)) {
11. char \*argv [3];
12. argv [0] = uevent\_helper;
13. argv [1] = (char \*)subsystem;
14. argv [2] = NULL;
15. retval = add\_uevent\_var(env, "HOME=/");
16. if (retval)
17. goto exit;
18. retval = add\_uevent\_var(env,
19. "PATH=/sbin:/bin:/usr/sbin:/usr/bin");
20. if (retval)
21. goto exit;
22. retval = call\_usermodehelper(argv[0], argv,
23. env->envp, UMH\_WAIT\_EXEC);
24. }
25. ...
26. }

复制代码

怎么样，够简洁吧，其实看实际的代码比这要郁闷地多，不过骨架清晰就行了。代码中的netlink message就不用多说了吧，给udev发通知用(有时间的话可以分析分析udev的代码)。本帖重点讨论后半段的if (uevent\_helper[0] && !kobj\_usermode\_filter(kobj))代码，这里的核心调用是call\_usermodehelper，这个函数最有意思的地方就在于在内核空间调用用户空间的程序，它的详细实现机制在书中已经讲得很多，这里就不再赘述了。call\_usermodehelper在kobject\_uevent\_env函数中要调用的用户空间程序由uevent\_helper[0]来指定，所以如果我们能控制这个uevent\_helper[0]，就能接收到设备加入系统移出系统等事件。那个if中的kobj\_usermode\_filter条件一般都会满足(除非这是个特别注意个人隐私的设备，那就不好说了，人家偷偷加入系统就是不想让你知道你也没有办法，但是udev还是能知道的)。  
  
下面看看uevent\_helper[0]来自何处：

1. <lib/kobject\_uevent.c>
2. char uevent\_helper[UEVENT\_HELPER\_PATH\_LEN] = CONFIG\_UEVENT\_HELPER\_PATH;

复制代码

貌似要通过内核配置来指定，我看了一下我系统中Linux目录下的.config文件，找到了下面这行：

1. <linux-3.1.6/.config>
2. #
3. # Generic Driver Options
4. #
5. CONFIG\_UEVENT\_HELPER\_PATH=""

复制代码

丫的，居然没指定，那么uevent\_helper[0]=""，这样的话我们在kobject\_uevent\_env函数中的那个if语句就没法满足了，看来要重新配置再编译内核了。不过想想sysfs这么强大，内核开发的那帮人好歹给留个用户空间的接口出来吧，一查看还真有：  
  
<kernel/ksysfs.c>

1. static ssize\_t uevent\_helper\_store(struct kobject \*kobj,
2. struct kobj\_attribute \*attr,
3. const char \*buf, size\_t count)
4. {
5. if (count+1 > UEVENT\_HELPER\_PATH\_LEN)
6. return -ENOENT;
7. memcpy(uevent\_helper, buf, count);
8. uevent\_helper[count] = '\0';
9. if (count && uevent\_helper[count-1] == '\n')
10. uevent\_helper[count-1] = '\0';
11. return count;
12. }

复制代码

尼玛，爽得简直是一塌糊涂，虽然俺那台马力强劲的机器编个全新的内核不过几分钟的事情，但是哪里有上面这个方法爽啊。马上进入到/sys/kernel目录下ls一把，截屏如下(点击放大)：  
  
uevent1.png   
  
有个uevent\_helper文件不是？那么我们现在可以把我们用户空间的程序给打进去了，我打算做个最简单的脚本/sbin/myhotplug，这个脚本只干一件事，在/home/dennis目录下生成一个hotplug文件：  
  
</sbin/myhotplug>

1. #!/bin/sh
2. cd /home/dennis
3. touch hotplug

复制代码

然后把这个脚本程序的文件名给打入到内核空间的uevent\_helper[0]上：

1. root@build-server:/sys/kernel# echo "/sbin/myhotplug" > uevent\_helper
2. root@build-server:/sys/kernel# cat uevent\_helper
3. /sbin/myhotplug

复制代码

好了，现在检查一下你的/home/dennis目录下面有没有hotplug这个文件，有的话就删掉，否则怎么知道是新生成的呢。现在，找个U盘插到你的电脑里，然后再看一下/home/dennis目录，有个hotplug文件对吧？如果你现在删除这个文件，再把U盘给拔了，你会再次发现这个文件。这意味着什么，意味着你可以轻而易举地捕捉到设备加入/移出系统等事件，如果你的脚本足够智能，那么你就会想到很多很有创意的玩法对吧？  
  
最后，对于PCI设备而言，Linux系统在启动过程中会扫描系统中所有PCI设备，对发现的每一个设备都会调用device\_add函数，正如你前面看到的那样，udev将会被通知，它负责找到对应的驱动模块并加载。当然，如果你愿意，你也可以去捕捉这些事件。