教你如何学习 linux 内核

毫不夸张地说，Kconfig 和 Makefile 是我们浏览内核代码时最为依仗的两个文件。基本上，Linux 内核中每一个目录下边都会有一个 Kconfig 文件和一个 Makefile 文件。对于一个希望能够在 Linux 内核的汪洋代码里看到一丝曙光的人来说，将它们放在怎么重要的地位都不过分。

我们去香港，通过海关的时候，总会有免费的地图和各种指南拿，有了它们在手里我们才不至于无头苍蝇般迷惘的行走在陌生的街道上。即使在内地出去旅游的时候一般来说也总是会首先找份地图，当然了，这时就是要去买了，拿是拿不到的，不同的地方有不同的特色， 只不过有的特色是服务，有的特色是索取。

Kconfig 和 Makefile 就是 Linux Kernel 迷宫里的地图。地图引导我们去认识一个城市，而 Kconfig 和 Makefile 则可以让我们了解一个 Kernel 目录下面的结构。我们每次浏览 kernel 寻找属于自己的那一段代码时，都应该首先看看目录下的这两个文件。

利用 Kconfig 和 Makefile 寻找目标代码

就像利用地图寻找目的地一样，我们需要利用 Kconfig 和 Makefile 来寻找所要研究的目标代码。比如我们打算研究 U 盘驱动的实现，因为 U 盘是一种 storage 设备，所以我们应该先进入到 drivers/usb/storage/目录。但是该目录下的文件很多，那么究竟哪些文件才是我们需要关注的?这时就有必要先去阅读 Kconfig 和 Makefile 文件。

对于 Kconfig 文件，我们可以看到下面的选项。

```
config USB_STORAGE_DATAFAB
bool "Datafab Compact Flash Reader support (EXPERIMENTAL)"
depends on USB_STORAGE && EXPERIMENTAL
help
Support for certain Datafab CompactFlash readers.
Datafab has a web page at <http://www.datafabusa.com/>.
```

显然，这个选项和我们的目的没有关系。首先它专门针对 Datafab 公司的产品，其次虽然 CompactFlash reader 是一种 flash 设备，但显然不是 U 盘。因为 drivers/usb/storage 目录下的代码是针对 usb mass storage 这一类设备，而不是针对某一种特定的设备。U 盘只是 usb mass storage 设备中的一种。再比如：

```
config USB_STORAGE_SDDR55
bool "SanDisk SDDR-55 SmartMedia support (EXPERIMENTAL)"
depends on USB_STORAGE && EXPERIMENTAL
help
Say Y here to include additional code to support the Sandisk SDDR-55
SmartMedia reader in the USB Mass Storage driver.
```

很显然这个选项是有关 SanDisk 产品的，并且针对的是 SM 卡，同样不是 U 盘，所以我们也不需要去关注。

事实上，很容易确定，只有选项 CONFIG_USB_STORAGE 才是我们真正需要关注的。

```
9 config USB_STORAGE
10 tristate "USB Mass Storage support"
11 depends on USB && SCSI
12 ---help---
13 Say Y here if you want to connect USB mass storage devices to your
14 computer's USB port. This is the driver you need for USB
15 floppy drives, USB hard disks, USB tape drives, USB CD-ROMs,
16 USB flash devices, and memory sticks, along with
17 similar devices. This driver may also be used for some cameras
18 and card readers.
19
20 This option depends on 'SCSI' support being enabled, but you
21 probably also need 'SCSI device support: SCSI disk support'

22 (BLK_DEV_SD) for most USB storage devices.

23

24 To compile this driver as a module, choose M here: the
25 module will be called usb-storage.
```

接下来阅读 Makefile 文件。

```
0 #
1 # Makefile for the USB Mass Storage device drivers.
2 #
3 # 15 Aug 2000, Christoph Hellwig
4 # Rewritten to use lists instead of if-statements.
5 #
6
7 EXTRA_CFLAGS := -Idrivers/scsi
8
9 obj-$(CONFIG_USB_STORAGE) += usb-storage.o
10
11 usb-storage-obj-$(CONFIG_USB_STORAGE_DEBUG) += debug.o
12 usb-storage-obj-$(CONFIG_USB_STORAGE_USBAT) += shuttle_usbat.o

13 usb-storage-obj-$(CONFIG_USB_STORAGE_SDDR09) += sddr09.o

14 usb-storage-obj-$(CONFIG_USB_STORAGE_SDDR55) += sddr55.o

15 usb-storage-obj-$(CONFIG_USB_STORAGE_FREECOM) += freecom.o
```

```
16 usb-storage-obj-$(CONFIG_USB_STORAGE_DPCM) += dpcm.o
17 usb-storage-obj-$(CONFIG_USB_STORAGE_ISD200) += isd200.o
18 usb-storage-obj-$(CONFIG_USB_STORAGE_DATAFAB) += datafab.o
19 usb-storage-obj-$(CONFIG_USB_STORAGE_JUMPSHOT) += jumpshot.o
20 usb-storage-obj-$(CONFIG_USB_STORAGE_ALAUDA) += alauda.o
21 usb-storage-obj-$(CONFIG_USB_STORAGE_ONETOUCH) += onetouch.o
22 usb-storage-obj-$(CONFIG_USB_STORAGE_KARMA) += karma.o
23
24 usb-storage-objs := scsiglue.o protocol.o transport.o usb.o \
25 initializers.o $(usb-storage-obj-y)
26
27 ifneq ($(CONFIG_USB_LIBUSUAL),)
28 obj-$(CONFIG_USB) += libusual.o
29 endif
```

前面通过 Kconfig 文件的分析，我们确定了只需要去关注 CONFIG_USB_STORAGE 选项。在 Makefile 文件里查找 CONFIG_USB_STORAGE，从第 9 行得知，该选项对应的模块为 usbstorage。因为 Kconfig 文件里的其他选项我们都不需要关注，所以 Makefile 的 11~22 行可以忽略。第 24 行意味着我们只需要关注 scsiglue.c、protocol.c、transport.c、usb.c、initializers.c 以及它们同名的.h 头文件。

Kconfig 和 Makefile 很好的帮助我们定位到了所要关注的目标，就像我们到一个陌生的地方要随身携带地图，当我们学习 Linux 内核时，也要谨记寻求 Kconfig 和 Makefile 的帮助。

透过现象看本质，兽兽门无非就是一些人体艺术展示。同样往本质里看过去，学习内核，就是学习内核的源代码，任何内核有关的书籍都是基于内核，而又不高于内核的。

既然要学习内核源码，就要经常对内核代码进行分析，而内核代码千千万，还前仆后继的不断往里加，这就让大部分人都有种雾里看花花不见的无助感。不过 不要怕，孔老夫子早就留给我们了应对之策：敏于事而慎于言，就有道而正焉，可谓好学也已。这就是说，做事要踏实才是好学生好同志，要遵循严谨的态

度，去理 解每一段代码的实现，多问多想多记。如果抱着走马观花，得过且过的态度，结果极有可能就是一边看一边丢，没有多大的收获。

假设全国房价上涨 1.5%，假设 80 后局长是农民子弟??，既然我们的人生充满了假设，那么我在这里假设你现在就迫不及待的希望研究内核中 USB 子系统的实现，应该没有意见吧?那好，下面就以 USB 子系统的实现分析为标本看看分析内核源码应该如何入手。

分析 README

内核中 USB 子系统的代码位于目录 drivers/usb，这个结论并不需要假设。于是我们进入到该目录，执行命令 ls，结果显示如下：

atm class core gadget host image misc mon serial storage Kconfig

Makefile README usb-skeleton.c

目录 drivers/usb 共包含有 10 个子目录和 4 个文件，usb-skeleton.c 是一个简单的 USB driver 的框架，感兴趣的可以去看看，目前来说，它还吸引不了我们的眼球。那么首先应该关注什么?如果迎面走来一个 ppmm，你会首先看脸、脚还是其 它?当然答案依据每个人的癖好会有所不同。不过这里的问题应该只有一个答案，那就是 Kconfig、Makefile、README。

README 里有关于这个目录下内容的一般性描述，它不是关键，只是帮助你了解。再说了，面对"read 我吧 read 我吧"这么热情奔放的呼唤，善良的我们是不可能无动于衷的，所以先来看看里面都有些什么内容。

```
23 Here is a list of what each subdirectory here is, and what is contained in
24 them.
25
26 core/ - This is for the core USB host code, including the
27 usbfs files and the hub class driver ("khubd").
28
29 host/ - This is for USB host controller drivers. This
30 includes UHCI, OHCI, EHCI, and others that might
31 be used with more specialized "embedded" systems.
32
33 gadget/ - This is for USB peripheral controller drivers and
34 the various gadget drivers which talk to them.
35
36
37 Individual USB driver directories. A new driver should be added to the

38 first subdirectory in the list below that it fits into.
39
40 image/ - This is for still image drivers, like scanners or
41 digital cameras.
```

42 input/ - This is for any driver that uses the input subsystem,
43 like keyboard, mice, touchscreens, tablets, etc.
44 media/ - This is for multimedia drivers, like video cameras,
45 radios, and any other drivers that talk to the v4l
46 subsystem.
47 net/ - This is for network drivers.
48 serial/ - This is for USB to serial drivers.

49 storage/ - This is for USB mass-storage drivers.
50 class/ - This is for all USB device drivers that do not fit
51 into any of the above categories, and work for a range
52 of USB Class specified devices.
53 misc/ - This is for all USB device drivers that do not fit
54 into any of the above categories.

这个 README 文件描述了前边使用 ls 命令列出的那 10 个文件夹的用途。那么什么是 USB Core?

Linux 内核开发者们，专门写了一些代码，负责实现一些核心的功能，为别的设备驱动程序提供服务，比如申请内存，比如实现一些所有的设备都会 需要的公共的函数，并美其名曰 USB Core。

时代总在发展，当年胖杨贵妃照样迷死唐明皇，而如今人们欣赏的则是林志玲这样的魔鬼身材。同样，早期的 Linux 内核，其结构并不是如今天这般有层 次感，远不像今天这般错落有致，那时候 drivers/usb/这个目录下边放了很多很多文件，USB Core 与其他各种设备的驱动程序的代码都堆砌

在这里，后来，怎奈世间万千的变幻，总爱把有情的人分两端。于是在 drivers/usb/目录下面出来了一个 core 目录，就专门放一些核心的代码，比如初始化整个 USB 系统，初始化 Root Hub，初始化主机控制器的代码，再后来甚至把主机控制器相关的代码也单独建了一个目录，叫 host 目录，这是因为 USB 主机控制器随着时代的发展，也开 始有了好几种，不再像刚开始那样只有一种，所以呢，设计者们把一些主机控制器公共的代码仍然留在 core 目录下，而一些各主机控制器单独的代码则移到 host 目录下面让负责各种主机控制器的人去维护。

那么 USB gadget 那?gadget 白了说就是配件的意思，主要就是一些内部运行 Linux 的嵌入式设备，比如 PDA，设备本身有 USB 设备控制器(USB Device Controller)，可以将 PC，也就是我们的主机作为 master 端，将这样的设备作为 slave 端和主机通过 USB 进行通信。从主机的观点来看， 主机系统的 USB 驱动程序控制插入其中的 USB 设备，而 USB gadget 的驱动程序控制外围设备如何作为一个 USB 设备和主机通_____信。比如，我们的嵌入式板子上支持 SD 卡，如果我们希望在将板子通过 USB 连接到 PC 之后，这个 SD 卡被模拟成 U 盘，那么就要通过 USB gadget 架构的驱动。

剩下的几个目录分门别类的放了各种 USB 设备的驱动，比如 U 盘的驱动在 storage 目录下，触摸屏和 USB 键盘鼠标的驱动在 input 目录下，等等。

我们响应了 README 的热情呼唤，它便给予了我们想要的，通过它我们了解了 USB 目录里的那些文件夹都有着什么样的角色。到现在为止，就只剩下内核的地图——Kconfig 与 Makefile 两个文件了。

有地图在手，对于在内核中游荡的我们来说，是件很愉悦的事情，不过，因为我们的目的是研究内核对 USB 子系统的实现，而不是特定设备或 host controller 的驱动，所以这里的定位很明显，USB Core 就是我们需要关注的对象，那么接下来就是要对 core 目录中的内容进行定位了。

分析 Kconfig 和 Makefile

进入到 drivers/usb/core 目录，执行命令 ls，结果显示如下：

```
Kconfig Makefile buffer.c config.c devices.c devio.c driver.c
endpoint.c file.c generic.c hcd-pci.c hcd.c hcd.h hub.c hub.h
inode.c message.c notify.c otg_whitelist.h quirks.c sysfs.c urb.c
usb.c usb.h
```

然后执行 wc 命令，如下所示。

```
# wc –l ./*
148 buffer.c
607 config.c
706 devices.c
1677 devio.c
1569 driver.c
357 endpoint.c
248 file.c
238 generic.c
1759 hcd.c
458 hcd.h
433 hcd-pci.c
3046 hub.c
195 hub.h
758 inode.c
```

```
144 Kconfig

21 Makefile

1732 message.c

68 notify.c

112 otg_whitelist.h

161 quirks.c

710 sysfs.c

589 urb.c

984 usb.c

160 usb.h

16880 total
```

drivers/usb/core 目录共包括 24 个文件，16880 行代码。core 不愧是 core，为大家默默的做这么多事。不过这么多文件里不一定都是我们所需要关注的，先拿咱们的地图来看看接下来该怎么走。

先看看 Kconfig 文件，可以看到下面的选项。

```
15 config USB_DEVICEFS

16 bool "USB device filesystem"

17 depends on USB

18 ---help---

19 If you say Y here (and to "/proc file system support" in the "File

20 systems" section, above), you will get a file /proc/bus/usb/devices

21 which lists the devices currently connected to your USB bus or

22 busses, and for every connected device a file named
```

```
23 "/proc/bus/usb/xxx/yyy", where xxx is the bus number and yyy the

24 device number; the latter files can be used by user space programs

25 to talk directly to the device. These files are "virtual", meaning

26 they are generated on the fly and not stored on the hard drive.

27

28 You may need to mount the usbfs file system to see the files, use

29 mount -t usbfs none /proc/bus/usb

30

31 For the format of the various /proc/bus/usb/ files, please read

32 <file:Documentation/usb/proc_usb_info.txt>.

33

34 Usbfs files can't handle Access Control Lists (ACL), which are the

35 default way to grant access to USB devices for untrusted users of a

36 desktop system. The usbfs functionality is replaced by real

37 device-nodes managed by udev. These nodes live in /dev/bus/usb and

38 are used by libusb.
```

选项 USB_DEVICEFS 与 usbfs 文件系统有关。usbfs 文件系统挂载在/proc/bus/usb 目录，显示了当前连接的所有 USB 设 备及总线的各种信息，每个连接的 USB 设备在其中都会有一个对应的文件进行描述。比如文件/proc/bus/usb/xxx/yyy，xxx 表示总_____线的 序号，yyy 表示设备所在总线的地址。不过不能够依赖它们来稳定地访问设备，因为同一设备两次连接对应的描述文件可能会不同，比如，第一次连接一个设备 时，它可能是 002/027，一段时间后再次连接，它可能就已经改变为 002/048。

就好比好不容易你暗恋的 mm 今天见你的时候对你抛了个媚眼，你心花怒放，赶快去买了 100 块彩票庆祝，到第二天再见到她的时候，她对你说你是谁啊，你悲痛欲绝的刮开那 100 块彩票，上面清一色的谢谢你。

因为 usbfs 文件系统并不属于 USB 子系统实现的核心部分，与之相关的代码我们可以不必关注。

```
74 config USB_SUSPEND

75 bool "USB selective suspend/resume and wakeup (EXPERIMENTAL)"

76 depends on USB && PM && EXPERIMENTAL

77 help

78 If you say Y here, you can use driver calls or the sysfs

79 "power/state" file to suspend or resume individual USB

80 peripherals.

81

82 Also, USB "remote wakeup" signaling is supported, whereby some

83 USB devices (like keyboards and network adapters) can wake up
84 their parent hub. That wakeup cascades up the USB tree, and

85 could wake the system from states like suspend-to-RAM.

86

87 If you are unsure about this, say N here.
```

这一项是有关 USB 设备的挂起和恢复。开发 USB 的人都是节电节能的好孩子，所以协议里就规定了，所有的设备都必须支持挂起状态，就是说为了达到节 电的目的，当设备在指定的时间内，如果没有发生总线传输，就要进入挂起状态。当它收到一个 non-idle 的信号时，就会被唤醒。节约用电从 USB 做起。 不过这个与主题也没太大关系，相关代码也可以不用关注了。

剩下的还有几项，不过似乎与咱们关系也不大，还是去看看 Makefile。

```
5 usbcore-objs := usb.o hub.o hcd.o urb.o message.o driver.o \

6 config.o file.o buffer.o sysfs.o endpoint.o \

7 devio.o notify.o generic.o quirks.o

8

9 ifeq ($(CONFIG_PCI),y)

10 usbcore-objs += hcd-pci.o
```

```
11 endif
12
13 ifeq ($(CONFIG_USB_DEVICEFS),y)
14 usbcore-objs += inode.o devices.o
15 endif
16
17 obj-$(CONFIG_USB) += usbcore.o
18
19 ifeq ($(CONFIG_USB_DEBUG),y)
20 EXTRA_CFLAGS += -DDEBUG
21 endif
```

Makefile 可比 Kconfig 简略多了，所以看起来也更亲切点，咱们总是拿的 money 越多越好，看的代码越少越好。这里之所以会出现 CONFIG_PCI，是因为通常 USB 的 Root Hub 包含在一个 PCI 设备中。hcd-pci 和 hcd 顾名而思义就知道是说主机控制器的，它们实现了主机控制器公共部分，按

协议里的说法它们就是 HCDI(HCD 的公共接口)，host 目录下则实现了各种不同的主机控制器。CONFIG_USB_DEVICEFS 前面的 Kconfig 文件里也见到了，关于 usbfs 的，与咱们的主题无关，inode.c 和 devices.c 两个文件也可以不用管了。

那么我们可以得出结论，为了理解内核对 USB 子系统的实现，我们需要研究

buffer.c、config.c、driver.c、 endpoint.c、file.c、generic.c、hcd.c

hcd.h、hub.c、message.c、notify.c、otg_whitelist.h、quirks.c、sysfs.c、urb.c 和 usb.c 文件。

这么看来，好像大都需要关注的样子，没有减轻多少压力，不过这里本身就是 USB Core 部分，是要做很多的事为咱们分忧的，所以多点也是可以理解的。

下面的分析，米卢教练说了，内容不重要，重要的是态度。就像韩局长对待日记的态度那样，严谨而细致。

只要你使用这样的态度开始分析内核，那么无论你选择内核的哪个部分作为切入点，比如 USB，比如进程管理，在花费相对不算很多的时间之后，你就会发 现你对内核的理解会上升到另外一个高度，一个抱着情景分析，抱着 0.1 内核完全注释，抱着各种各样的内核书籍翻来覆去的看很多遍又忘很多遍都无法达到的高 度。请相信我!让我们在 Linux 社区里发出号召：学习内核源码，从学习韩局长开始!

态度决定一切：从初始化函数开始

任小强们说房价高涨从现在开始，股评家们说牛市从 5000 点开始。他们的开始需要我们的钱袋，我们的开始只需要一台电脑，最好再有一杯茶，伴着几支小曲儿，不盯着钱总是会比较惬意的。生容易，活容易，生活不容易，因为总要盯着钱。

有了地图 Kconfig 和 Makefile，我们可以在庞大复杂的内核代码中定位以及缩小了目标代码的范围。那么现在，为了研究内核对 USB 子系统的实现，我们还需要在目标代码中找到一个突破口，这个突破口就是 USB 子系统的初始化代码。

针对某个子系统或某个驱动，内核使用 subsys_initcall 或 module_init 宏指定初始化函数。在 drivers/usb/core/usb.c 文件中，我们可以发现下面的代码。

940 subsys_initcall(usb_init);

941 module_exit(usb_exit);

我们看到一个 subsys_initcall，它也是一个宏，我们可以把它理解为 module_init，只不过因为这部分代码比较核心，开发者们 把它看作一个子系统，而不仅仅是一个模块。这也很好理解，usbcore 这个模块它代表的不是某一个设备，而是所有 USB 设备赖以生存的模块，Linux 中，像这样一个类别的设备驱动被归结为一个子系统。比如 PCI 子系统，比如 SCSI 子系统，基本上，drivers/目录下面第一层的每个目录都算一个子 系统，因为它们代表了一类设备。

subsys_initcall(usb_init)的意思就是告诉我们 usb_init 是 USB 子系统真正的初始化函数，而 usb_exit() 将是整个 USB 子系统的结束时的清理函数。于是为了研究 USB 子系统在内核中的实现，

我们需要从 usb_init 函数开始看起。

```
865 static int __init usb_init(void)

866 {

867 int retval;

868 if (nousb) {

869 pr_info("%s: USB support disabled\n", usbcore_name);

870 return 0;
```

```
871 }
872
873 retval = ksuspend_usb_init();
874 if (retval)
875 goto out;
876 retval = bus_register(&usb_bus_type);
877 if (retval)
878 goto bus_register_failed;
879 retval = usb_host_init();
880 if (retval)
881 goto host_init_failed;
882 retval = usb_major_init();
883 if (retval)
884 goto major_init_failed;
885 retval = usb_register(&usbfs_driver);
886 if (retval)
887 goto driver_register_failed;
888 retval = usb_devio_init();
889 if (retval)
890 goto usb_devio_init_failed;
891 retval = usbfs_init();
892 if (retval)
```

```
893 goto fs_init_failed;

894 retval = usb_hub_init();

895 if (retval)

896 goto hub_init_failed;

897 retval = usb_register_device_driver(&usb_generic_driver, THIS_MODULE);

898 if (!retval)

899 goto out;

900

901 usb_hub_cleanup();

902 hub_init_failed:

903 usbfs_cleanup();

904 fs_init_failed:

905 usb_devio_cleanup();

906 usb_devio_init_failed:

907 usb_deregister(&usbfs_driver);

908 driver_register_failed:

909 usb_major_cleanup();

910 major_init_failed:

911 usb_host_cleanup();

912 host_init_failed:

913 bus_unregister(&usb_bus_type);

914 bus_register_failed:
```

```
915 ksuspend_usb_cleanup();

916 out:

917 return retval;

918 }
```

(1)__init 标记。

关于 usb_init，第一个问题是，第 865 行的__init 标记具有什么意义?

写过驱动的应该不会陌生，它对内核来说就是一种暗示，表明这个函数仅在初始化期间使用，在模块被装载之后，它占用的资源就会释放掉用作它处。它的暗 示你懂，可你的暗示，她却不懂或者懂装不懂，多么让人感伤。它在自己短暂的一生中一直从事繁重的工作，吃的是草吐出的是牛奶，留下的是整个 USB 子系统的 繁荣。

受这种精神所感染，我觉得_____有必要为它说的更多些。__init 的定义在 include/linux/init.h 文件里 43 #define __init __attribute__ ((__section__ (".init.text")))

好像这里引出了更多的疑问，__attribute__是什么?Linux 内核代码使用了大量的 GNU C 扩展，以至于 GNU C 成为能够编译内核的唯一编译器，GNU C 的这些扩展对代码优化、目标代码布局、安全检查等方面也提供了很强的支持。而__attribute__就是这些扩展中的一个，它主要被用来声明一些特殊的属性，这些属性主要被用来指示编译器进行特定方面的优化和更仔细的代码检查。GNU C 支持十几个属性，section 是其中的一个，我们查看 GCC 的手册可以看到下面的描述

`section ("section-name")'

Normally, the compiler places the code it generates in the `text' section. Sometimes, however, you need additional sections, or you need certain particular functions to appear in special sections.The `section' attribute specifies that a function lives in a particular section. For example, the declaration:extern void foobar (void) __attribute__ ((section ("bar")));puts the function `foobar' in the `bar' section.Some file formats do not support arbitrary sections so the `section' attribute is not available on all platforms. If you need to map the entire contents of a module to a particular section, consider using the facilities of the linker instead.

通常编译器将函数放在.text 节，变量放在.data 或.bss 节，使用 section 属性，可以让编译器将函数或变量放在指定的节中。那么前面 对__init 的定义便表示将它修饰的代码放在.init.text 节。连接器可以把相同节的代码或数据安排在一起，比如__init 修饰的所有代码都会 被放在.init.text 节里，初始化结束后就可以释放这部分内存。

问题可以到此为止，也可以更深入，即内核又是如何调用到这些__init 修饰的初始化函数?要回答这个问题，还需要回顾一下 subsys_initcall 宏，它也在 include/linux/init.h 里定义

125 #define subsys_initcall(fn) __define_initcall("4",fn,4)

这里又出现了一个宏__define_initcall，它用于将指定的函数指针 fn 放到 initcall.init 节里 而对于具体的 subsys_initcall 宏，则是把 fn 放到.initcall.init 的子节.initcall4.init 里。要弄清楚.initcall.init、.init.text 和.initcall4.init 这样的东东，我们还需要了解一点内核可执行文件相关的概念。

内核可执行文件由许多链接在一起的对象文件组成。对象文件有许多节，如文本、数据、init 数据、bass 等等。这些对象文件都是_____由一个称为链接器 脚本的文件链接并装入的。这个链接器脚本的功能是将输入对象文件的各节映射到输出文件中;换句话说，它将所有输入对象文件都链接到单一的可执行文件中，将 该可执行文件的各节装入到指定地址处。 vmlinux.lds 是存在于 arch// 目录中的内核链接器脚本，它负责链接内核的各个节并将它们装入内存中特定偏移量处。

我可以负责任的告诉你，要看懂 vmlinux.lds 这个文件是需要一番功夫的，不过大家都是聪明人，聪明人做聪明事，所以你需要做的只是搜索 initcall.init，然后便会看到似曾相识的内容

```
__inicall_start = .;

.initcall.init : AT(ADDR(.initcall.init) – 0xC0000000)
*(.initcall1.init)

*(.initcall2.init)

*(.initcall3.init)

*(.initcall4.init)

*(.initcall5.init)

*(.initcall6.init)

*(.initcall7.init)

}

__initcall_end = .;
```

这里的__initcall_start 指向.initcall.init 节的开始，__initcall_end 指向它的结尾。而.initcall.init 节又被分为了 7 个子节，分别是

```
.initcall1.init

.initcall2.init

.initcall3.init
```

```
.initcall4.init

.initcall5.init

.initcall6.init

.initcall7.init
```

我们的 subsys_initcall 宏便是将指定的函数指针放在了.initcall4.init 子节。其它的比如 core_initcall 将函数指针放在.initcall1.init 子节，device_initcall 将函数指针放在了.initcall6.init 子节等等，都可以从 include/linux/init.h 文件找到它们的定义。各个字节的顺序是确定的，即先调用.initcall1.init 中的函数指针再调 用.initcall2.init 中的函数指针，等等。__init 修饰的初始化函数在内核初始化过程中调用的顺序和.initcall.init 节里函 数指针的顺序有关，不同的初始化函数被放在不同的子节中，因此也就决定了它们的调用顺序。

至于实际执行函数调用的地方，就在/init/main.c 文件里，内核的初始化么，不在那里还能在哪里，里面的 do_initcalls 函数会直接用到这里的__initcall_start、__initcall_end 来进行判断。

(2)模块参数。

关于 usb_init 函数，第二个问题是，第 868 行的 nousb 表示什么?

知道 C 语言的人都会知道 nousb 是一个标志，只是不同的标志有不一样的精彩，这里的 nousb 是用来让我们在启动内核的时候通过内核参数去掉 USB 子系统的，Linux 社会是一个很人性化的世界，它不会去逼迫我们接受 USB，一切都只关乎我们自己的需要。不过我想我们一般来说是不会去指定 nousb 的吧。如果你真的指定了 nousb，那它就只会幽怨的说一句“USB support disabled”，然后退出 usb_init。

nousb 在 drivers/usb/core/usb.c 文件中定义为：

static int nousb; /* Disable USB when built into kernel image */

module_param_named(autosuspend, usb_autosuspend_delay, int, 0644);

MODULE_PARM_DESC(autosuspend, "default autosuspend delay");

从中可知 nousb 是个模块参数。关于模块参数，我们都知道可以在加载模块的时候可以指定，但是如何在内核启动的时候指定?

打开系统的 grub 文件，然后找到 kernel 行，比如：

kernel /boot/vmlinuz-2.6.18-kdb root=/dev/sda1 ro splash=silent vga=0x314

其中的 root，splash，vga 等都表示内核参数。当某一模块被编译进内核的时候，它的模块参数便需要在 kernel 行来指定，格式为“模块名.参数=值”，比如：

modprobe usbcore autosuspend=2

对应到 kernel 行，即为 ：

usbcore.autosuspend=2

通过命令“modinfo -p ${modulename}”可以得知一个模块有哪些参数可以使用。同时，对于已经加载到内核里的模块，它们的模块参数会列举在/sys/module /${modulename}/parameters/目录下面，可以使用“echo -n ${value} > /sys/module/${modulename}/parameters/$

{parm}”这样的命令去修改。

(3)可变参数宏。

关于 usb_init 函数，第三个问题是，pr_info 如何实现与使用?pr_info 只是一个打印信息的可辨参数宏，printk 的变体，在 include/linux/kernel.h 里定义：

242 #define pr_info(fmt,arg...) \

243 printk(KERN_INFO fmt,##arg)

99 年的 ISO C 标准里规定了可变参数宏，和函数语法类似，比如

#define debug(format, ...) fprintf (stderr, format, __VA_ARGS__)里面的“…”就表示可变参数，调用时，它们就会替代宏体里的__VA_ARGS__。GCC 总是会显得特立独行一些，它支持更复杂的形式，可以给可变参数取个名字，比如#define debug(format, args...) fprintf (stderr, format, args)

有了名字总是会容易交流一些。是不是与 pr_info 比较接近了?除了‘##’，它主要是针对空参数的情况。既然说是可变参数，那传递空参数也总是可以的，空即是多，多即是空，股市里的哲理这里同样也是适合的。如果没有‘##’，传递空参数的时候，比如 debug ("A message");展开后，里面的字符串后面会多个多余的逗号。这个逗号你应该不会喜欢，而‘##’则会使预处理器去掉这个多余的逗号。

关于 usb_init 函数，上面的三个问题之外，余下的代码分别完成 usb 各部分的初始化，接下来就需要围绕它们分别进行深入分析。因为这里只是演示如何入手分析，展示的只是一种态度，所以具体的深入分析就免了吧。

对于学习来说，无论是在学校的课堂学习，还是这里说的内核学习，效果好或者坏，最主要取决于两个方面——方法论和心理。注意，我无视了智商的差异，这玩意儿玄之又玄，岔开了说，属于迷信的范畴。

前面又是 Kernel 地图，又是如何入手，说的都是方法论的问题，那么这里要面对的就主要是心理上的问题。

而心理上的问题主要有两个，一个是盲目，就是在能够熟练适用 Linux 之前，对 Linux 为何物还说不出个道道来，就迫不及待的盲目的去研究内核的 源代码。这一部分人会觉得既然是学习内核，那么耗费时间在熟悉 Linux 的基本操作上纯粹是浪费宝贵的时间和感情。不过这样虽然很有韩峰同志的热情和干劲儿，但明显走入了一种心理误区。重述 Linus 的那句话：要先会使用它。

第二个就是恐惧。人类进化这么多年，面对复杂的物体和事情还是总会有天生的惧怕感，体现在内核学习上面就是：那么庞大复杂的内核代码，让人面对起来该情何以堪啊!

有了这种恐惧无力感存在，心理上就会去排斥面对接触内核源码，宁愿去抱着情景分析，搜集各种各样五花八门的内核书籍放在那里屯着，看了又忘，忘了又看，也不大情愿去认真细致得浏览源码。

这个时候，我们在心理上是脆弱得，我们忘记了芙蓉姐姐，工行女之所以红起来，不是她们有多好，而是因为她们得心理足够坚强。是的，除了向韩局长学习态度，我们还要向涌现出来的无数个芙蓉姐姐和工行女学习坚强的心理。

有必要再强调一次，学习内核，就是学习内核的源代码，任何内核有关的书籍都是基于内核，而又不高于内核的。内核源码本身就是最好的参考资料，其他任何经典或非经典的书最多只是起到个辅助作用，不能也不应该取代内核代码在我们学习过程中的主导地位。

“世界上最缺的不是金钱，而是资源。”当我在一份报纸上看到这句大大标题时，我的第一反应是——作者一定是个自然环保主义者，然后我在羞愧得反省自身的同时油然生出一股对这样的无产主义理想者无比崇敬的情绪来。于是，我继续往下看，“因此在 XXX 还未正式面市之时，前来咨询的客户已经不少，这些有眼光的购房者明白，谁能在目前最好的购房机会下最大化地占有绝版资 源，谁就掌控了未来财富流向。”(为了避免做广告的嫌疑，请允许我使用 XXX 代替该楼盘的名字。)顿时，我悟道了!

其实，韩峰同志已经在日记里告诉了我们资源的重要性，因此我们在学习韩峰同志严谨细致的态度同时，还要领悟他对资源的灵活运用。只有在以内核源码为中心，坚持各种学习资源的长期建设不动摇，才能达到韩局长那样的高度，俯视 Linux 内核世界里的人生百态。

注意，这个观点与前面所说的学习效果主要取决于方法论和心理两个方面并不矛盾，它们属于不同层次上的问题。

内核文档

内核代码中包含有大量的文档，这些文档对于学习理解内核有着不可估量的价值，记住，在任何时候，它们在我们心目中的地位都应该高于那些各式的内核参考书。下面是一些内核新人所应该阅读的文档。

README

这个文件首先简单介绍了 Linux 内核的背景，然后描述了如何配置和编译内核，最后还告诉我们出现问题时应该怎么办。

Documentation/Changes 这个文件给出了用来编译和使用内核所需要的最小软件包列表。

Documentation/CodingStyle 这个文件描述了内核首选的编码风格，所有代码都应该遵守里面定义的规范。

Documentation/SubmittingPatches

Documentation/SubmittingDrivers

Documentation/SubmitChecklist

这三个文件都是描述如何提交代码的，其中 SubmittingPatches 给出创建和提交补丁的过程，

SubmittingDrivers 描述了如何将 设备驱动提交给 2.4、2.6 等不同版本的内核树，SubmitChecklist 则描述了提交代码之前需要 check 自己的代码应该遵守的某些事项。

Documentation/stable_api_nonsense.txt 这个文件解释了为什么内核没有一个稳定的内部 API(到用户空间的接口——系统调用——是稳定的)，它对于理解 Linux 的开发哲学至关重要，对于将开发平台从其他操作系统转移到 Linux 的开发者来说也很重要。

Documentation/stable_kernel_rules.txt 解释了稳定版内核(stable releases)发布的规则，以及如何将补丁提交给这些版本。

Documentation/SecurityBugs

内核开发者对安全性问题非常关注，如果你认为自己发现了这样的问题，可以根据这个文件中给出的联系方式提交 bug，以便能够尽可能快的解决这个问题。

Documentation/kernel-docs.txt 这个文件列举了很多内核相关的文档和书籍，里面不乏经典之作。

Documentation/applying-patches.txt 这个文件回答了如何为内核打补丁。

Documentation/bug-hunting 这个文件是有关寻找、提交、修正 bug 的。

Documentation/HOWTO 这个文件将指导你如何成为一名内核开发者，并且学会如何同内核开发社区合作。它尽可能不包括任何关于内核编程的技术细节，但会给你指引一条获得这些知识的正确途径。

经典书籍

待到山花烂漫时，还是那些经典在微笑。

有关内核的书籍可以用汗牛充栋来形容，不过只有一些经典的神作经住了考验。首先是 5 本久经考验的神作(个人概括为“2+1+2”，第一个 2 是指 2 本全面讲 解内核的书，中间的 1 指 1 本讲解驱动开发的书，后面的 2 则指 2 本有关内核具体子系统的书，你是否想到了某某广告里三个人突然站起单臂齐举高呼“1 比 1 比 1”的场景?)。

《Linux 内核设计与实现》

简称 LKD，从入门开始，介绍了诸如进程管理、系统调用、中断和中断处理程序、内核同步、时

间管理、内存管理、地址空间、调试技术等方面，内容比较浅显易懂，个人认为是内核新人首先必

读的书籍。新人得有此书，足矣!

《深入理解 Linux 内核》

简称 ULK，相比于 LKD 的内容不够深入、覆盖面不广，ULK 要深入全面得多。

前面这两本，一本提纲挈领，一本全面深入。

《Linux 设备驱动程序》

简称 LDD，驱动开发者都要人手一本了。

《深入理解 Linux 虚拟内存管理》

简称 LVMM，是一本介绍 Linux 虚拟内存管理机制的书。如果你希望深入的研究 Linux 的内存管理

子系统，仔细的研读这本书无疑是最好的选择。

《深入理解 LINUX 网络内幕》

一本讲解网络子系统实现的书，通过这本书，我们可以了解到 Linux 内核是如何实现复杂的网络功能的。(忘了声明下，我这列出来的书名是中文的，但是并不代表我建议大家去看他们的中文版，其中有的翻译的实在太??了，呵呵)

这 5 本书各有侧重，正如下面的图所展示的那样，恰好代表了个人一直主张的内核学习方法：首先通过 LKD 或 ULK 了解内核的设计实现特点，对内核有个整体全局的认识和理解，然后可分为两个岔路，如果从事驱动开发，则钻研 LDD，如果希望对内核不是泛泛而谈而是有更深入的理解，则可以选择一个自己感兴趣的子系统，仔细分析它的代码，不懂的地方就通过社区、邮件列表或者直接发 Email 给 maintainer 请教等途径弄懂，切勿得过且过，这样分析下来，对同步、中断等等内核的很多机制也同样会非常了解，俗话说的一通则百通就是这个道理。当然，如果你选择研究的是内存管理或者网络，则可以有上面的两本书可以学习，如果是其他子系统可能就没有这么好的运气了。

内核社区

最近几年，社区网站非常的热火，不过此社区非彼社区。

Linux 最大的一个优势就是它有一个紧密团结了众多使用者和开发者的社区，它的目标就是提供尽善尽美的内核。内核社区的中心是内核邮件列表(Linux Kernel Mailing List，LKML)，我们可以在 http://vger.kernel.org/vger-lists.html#linux-kernel 上面看到订阅这个邮件列表的细节。

内核邮件列表的流量很大，每天都有几百条消息，这里是大牛们的战场，小牛们的天堂，任何一个内核开发者都可以从中受益非浅。除了 LKML，大多数子系统也有自己独立的邮件列表来协调各自的开发工作，比如 USB 子系统的邮件列表可以在 http://www.linux-usb.org/mailing.html 上面订阅。

其他网络资源

除了内核邮件列表，还有很多其他的论坛或网站值得我们经常关注。我们要知道，网络上不仅有兽兽和凤姐，也不仅有犀利哥和韩局长。http://www.kernel.org/ 可以通过这个网站上下载内核的源代码和补丁、跟踪内核 bug 等。http://kerneltrap.org Linux 和 BSD 内核的技术新闻。如果没时间跟踪 LKML，那么经常浏览 kerneltrap 是个好主意。http://lwn.net/ Linux weekly news，创建于 1997 年底的一

个 Linux 新闻站点。http://zh-kernel.org/mailman/listinfo/linux-kernel 这是内核开发的中文邮件列表，里面活跃着很多内核开发领域的华人，比如 Herbert Xu,、Mingming Cao、Bryan Wu 等。