

努力成为 linux kernel hacker 的人李万鹏原创作品，为梦而战。转载请标明出处

<http://blog.csdn.net/woshixingaaa/archive/2011/06/02/6462065.aspx>

本文档讲解一下驱动中常用的宏，下边一个一个来说，先声明我使用的内核是 Linux2.6.30.4。

Linux 在 arch/\$(ARCH)/kernel/vmlinux.lds 中定义了.init 段，当内核启动完毕，这个段中的内存会被释放掉供其他使用，vmlinux.lds 部分内容如下：

```
1. OUTPUT_ARCH(arm)
2. ENTRY(stext)
3. jiffies = jiffies_64;
4. SECTIONS
5. {
6.   . = 0xC0000000 + 0x00008000;
7.   .text.head : {
8.     _stext = .;
9.     _sinittext = .;
10.    *(.text.head)
11.  }
12.  .init : { /* Init code and data */
13.    *(.init.text) *(.cpuinit.text) *(.meminit.text)
14.    _einittext = .;
15.    __proc_info_begin = .;
16.    *(.proc.info.init)
17.    __proc_info_end = .;
18.    __arch_info_begin = .;
19.    *(.arch.info.init)
20.    __arch_info_end = .;
21.    __tagtable_begin = .;
22.    *(.taglist.init)
23.    __tagtable_end = .;
24.    . = ALIGN(16);
25.    __setup_start = .;
26.    *(.init.setup)
27.    __setup_end = .;
28.    __early_begin = .;
29.    *(.early_param.init)
30.    __early_end = .;
31.    __initcall_start = .;
32.    *(.initcallearly.init) __early_initcall_end = .;
    *(.initcall0.init) *(.initcall0s.init) *(.initcall1.init) *(.initcall1s.init)
    *(.initcall2.init) *(.initcall2s.init) *(.initcall3.init) *(.initcall3s.init)
    *(.initcall4.init) *(.initcall4s.init) *(.initcall5.init) *(.initcall5s.init)
    *(.initcall6.init) *(.initcall6s.init) *(.initcall7.init) *(.initcall7s.init)
```

```

33. __initcall_end = .;
34. __con_initcall_start = .;
35. *(.con_initcall.init)
36. __con_initcall_end = .;
37. __security_initcall_start = .;
38. *(.security_initcall.init)
39. __security_initcall_end = .;
40. . = ALIGN(32);
41. __initramfs_start = .;
42. usr/built-in.o(.init.ramfs)
43. __initramfs_end = .;
44. . = ALIGN(4096);
45. __per_cpu_load = .;
46. __per_cpu_start = .;
47. *(.data.percpu.page_aligned)
48. *(.data.percpu)
49. *(.data.percpu.shared_aligned)
50. __per_cpu_end = .;
51. __init_begin = _stext;
52. *(.init.data) *(.cpuinit.data) *(.cpuinit.rodata) *(.meminit.data) *(.meminit.rodata)
53. . = ALIGN(4096);
54. __init_end = .;
55. }
56. /DISCARD/ : { /* Exit code and data */
57. *(.exit.text) *(.cpuexit.text) *(.memexit.text)
58. *(.exit.data) *(.cpuexit.data) *(.cpuexit.rodata) *(.memexit.data) *(.memexit.rodata)
59. *(.exitcall.exit)
60. *(.ARM.exidx.exit.text)
61. *(.ARM.exstab.exit.text)
62. }
63. .text : { /* Real text segment */
64. __text = .; /* Text and read-only data */
65. __exception_text_start = .;
66. *(.exception.text)
67. __exception_text_end = .;
68. . = ALIGN(8); *(.text.hot) *(.text) *(.ref.text) *(.devinit.text) *(.devexit.text) *(.text.unlikely)
69. . = ALIGN(8); __sched_text_start = .; *(.sched.text) __sched_text_end = .;
70. . = ALIGN(8); __lock_text_start = .; *(.spinlock.text) __lock_text_end = .;
71. . = ALIGN(8); __kprobes_text_start = .; *(.kprobes.text) __kprobes_text_end = .;
72. *(.fixup)
73. *(.gnu.warning)
74. *(.rodata)
75. *(.rodata.*)
76. *(.glue_7)
77. *(.glue_7t)

```

```

78. *(.got) /* Global offset table */
79. }
80. . = ALIGN((4096)); .rodata : AT(ADDR(.rodata) - 0) { __start_rodata = .; *(.rodata) *(.rodata.*) *(
    vermagic) *(__markers_strings) *(__tracepoints_strings) } .rodata1 : AT(ADDR(.rodata1) - 0) { *(.ro
    data1) } .pci_fixup : AT(ADDR(.pci_fixup) - 0) { __start_pci_fixups_early = .; *(.pci_fixup_early) __en
    d_pci_fixups_early = .; __start_pci_fixups_header = .; *(.pci_fixup_header) __end_pci_fixups_header
    = .; __start_pci_fixups_final = .; *(.pci_fixup_final) __end_pci_fixups_final = .; __start_pci_fixups_ena
    ble = .; *(.pci_fixup_enable) __end_pci_fixups_enable = .; __start_pci_fixups_resume = .; *(.pci_fixu
    p_resume) __end_pci_fixups_resume = .; __start_pci_fixups_resume_early = .; *(.pci_fixup_resume_
    early) __end_pci_fixups_resume_early = .; __start_pci_fixups_suspend = .; *(.pci_fixup_suspend) __
    end_pci_fixups_suspend = .; } .builtin_fw : AT(ADDR(.builtin_fw) - 0) { __start_builtin_fw = .; *(.built
    in_fw) __end_builtin_fw = .; } .rio_route : AT(ADDR(.rio_route) - 0) { __start_rio_route_ops = .; *(.rio
    _route_ops) __end_rio_route_ops = .; } __ksymtab : AT(ADDR(__ksymtab) - 0) { __start__ksymtab
    = .; *(__ksymtab) __stop__ksymtab = .; } __ksymtab_gpl : AT(ADDR(__ksymtab_gpl) - 0) { __start
    __ksymtab_gpl = .; *(__ksymtab_gpl) __stop__ksymtab_gpl = .; } __ksymtab_unused : AT(ADDR(__
    ksymtab_unused) - 0) { __start__ksymtab_unused = .; *(__ksymtab_unused) __stop__ksymtab_un
    used = .; } __ksymtab_unused_gpl : AT(ADDR(__ksymtab_unused_gpl) - 0) { __start__ksymtab_un
    used_gpl = .; *(__ksymtab_unused_gpl) __stop__ksymtab_unused_gpl = .; } __ksymtab_gpl_future
    : AT(ADDR(__ksymtab_gpl_future) - 0) { __start__ksymtab_gpl_future = .; *(__ksymtab_gpl_future)
    __stop__ksymtab_gpl_future = .; } __kcrctab : AT(ADDR(__kcrctab) - 0) { __start__kcrctab = .; *(
    kcrctab) __stop__kcrctab = .; } __kcrctab_gpl : AT(ADDR(__kcrctab_gpl) - 0) { __start__kcrctab_gp
    l = .; *(__kcrctab_gpl) __stop__kcrctab_gpl = .; } __kcrctab_unused : AT(ADDR(__kcrctab_unused) -
    0) { __start__kcrctab_unused = .; *(__kcrctab_unused) __stop__kcrctab_unused = .; } __kcrctab_u
    nused_gpl : AT(ADDR(__kcrctab_unused_gpl) - 0) { __start__kcrctab_unused_gpl = .; *(__kcrctab_u
    nused_gpl) __stop__kcrctab_unused_gpl = .; } __kcrctab_gpl_future : AT(ADDR(__kcrctab_gpl_futur
    e) - 0) { __start__kcrctab_gpl_future = .; *(__kcrctab_gpl_future) __stop__kcrctab_gpl_future = .; }
    __ksymtab_strings : AT(ADDR(__ksymtab_strings) - 0) { *(__ksymtab_strings) } __init_rodata : AT(A
    DDR(__init_rodata) - 0) { *(.ref.rodata) *(.devinit.rodata) *(.devexit.rodata) } __param : AT(ADDR(_
    _param) - 0) { __start__param = .; *(__param) __stop__param = .; . = ALIGN((4096)); __end_rodat
    a = .; } . = ALIGN((4096));
81. _etext = .; /* End of text and rodata section */
82. /*
83.  * Stack unwinding tables
84.  */
85. . = ALIGN(8);
86. .ARM.unwind_idx : {
87.     __start_unwind_idx = .;
88.     *(.ARM.exidx*)
89.     __stop_unwind_idx = .;
90. }
91. .ARM.unwind_tab : {
92.     __start_unwind_tab = .;
93.     *(.ARM.exstab*)
94.     __stop_unwind_tab = .;
95. }
96. . = ALIGN(8192);
97. __data_loc = .;
98. .data : AT(__data_loc) {
99.     __data = .; /* address in memory */

```

```

100. /*
101.  * first, the init task union, aligned
102.  * to an 8192 byte boundary.
103.  */
104. *(.data.init_task)
105. . = ALIGN(4096);
106. __nosave_begin = .;
107. *(.data.nosave)
108. . = ALIGN(4096);
109. __nosave_end = .;
110. /*
111.  * then the cacheline aligned data
112.  */
113. . = ALIGN(32);
114. *(.data.cacheline_aligned)
115. /*
116.  * The exception fixup table (might need resorting at runtime)
117.  */
118. . = ALIGN(32);
119. __start__ex_table = .;
120. *(__ex_table)
121. __stop__ex_table = .;
122. /*
123.  * and the usual data section
124.  */
125. *(.data) *(.ref.data) *(.devinit.data) *(.devexit.data) . = ALIGN(8); __start__markers = .; *(__ma
    rkers) __stop__markers = .; . = ALIGN(32); __start__tracepoints = .; *(__tracepoints) __stop__trac
    epoints = .; . = ALIGN(8); __start__verbose = .; *(__verbose) __stop__verbose = .;
126. CONSTRUCTORS
127. _edata = .;
128. }
129. _edata_loc = __data_loc + SIZEOF(.data);
130. .bss : {
131.  __bss_start = .; /* BSS */
132.  *(.bss)
133.  *(COMMON)
134.  _end = .;
135. }
136. /* Stabs debugging sections. */
137. .stab 0 : { *(.stab) }
138. .stabstr 0 : { *(.stabstr) }
139. .stab.excl 0 : { *(.stab.excl) }
140. .stab.exclstr 0 : { *(.stab.exclstr) }
141. .stab.index 0 : { *(.stab.index) }
142. .stab.indexstr 0 : { *(.stab.indexstr) }

```

```

143. .comment 0 : { *(.comment) }
144.}
145./*
146. * These must never be empty
147. * If you have to comment these two assert statements out, your
148. * binutils is too old (for other reasons as well)
149. */
150.ASSERT((__proc_info_end - __proc_info_begin), "missing CPU support")
151.ASSERT((__arch_info_end - __arch_info_begin), "no machine record defined")

```

—：

只要你写过模块程序 hello world 就对 `__init`, `__exit` 不会陌生，他们定义在 `include/linux/init.h` 中：

```

1. #define __init __section(.init.text) __cold notrace
2. #define __exit __section(.exit.text) __exitused __cold

```

`__cold` 在 `include/linux/compiler-gcc4.h` 中定义：

```

1. #define __cold __attribute__((__cold__))

```

所以 `#define __init __section(.init.text) __cold notrace` 等价于 `#define __init __attribute__((__section(.init.text)))`

`__attribute__` 是一个 GNU C 扩展，它主要用来声明一些特殊的属性，这些属性主要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 支持几十个属性，`section` 是其中的一个。通常编译器将函数放在 `.text` 节，变量放在 `.data` 节或 `.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。一般在程序的结尾都会有一句，例如 `module_init(hello_init);` `hello_init` 就是那个被 `__init` 修饰的模块初始化函数，在 `insmod` 的时候会调用 `module_init` 中的函数。`__exit` 是在模块卸载时相应的内存释放。

下边说一下 `__initdata`，这个我在 DMA 的源码中看到过，定义在 `include/linux/init.h` 中：

```

1. #define __initdata __section(.init.data)


```

看上边的 `vmlinux.lds`，`__initdata` 段也在 `.init` 段中，说明初始化后他所修饰的函数占用的内存后会被释放掉。

在阅读 RTC 源码时遇到的 `__devinit`，`__devexit`。在 `include/linux/init.h` 中定义：

1. `#define __devinit __section(.devinit.text) __cold`
2. `#define __devexit __section(.devexit.text) __exitused __cold`

整个 .init 段释放 memory 的大小会在系统启动过程中打印出来：



二：

subsys\_initcall 定义在 include/linux/init.h 中，定义如下：

1. `#define subsys_initcall(fn) __define_initcall("4",fn,4)`

这里出现了一个宏 \_\_define\_initcall，他用于将指定的函数指针放到 initcall.init 节里，而对于具体的 subsys\_initcall 宏，则是把 fn 放到 .initcall.init 的子节 .initcall.init 里。看上边 vmlinux.lds 这一部分：

1. `__initcall_start = .;`
2. `*(.initcallearly.init) __early_initcall_end = .; *(.initcall0.init) *(.initcall0s.init) *(.initcall1.init) *(.initcall1s.init) *(.initcall2.init) *(.initcall2s.init) *(.initcall3.init) *(.initcall3s.init) *(.initcall4.init) *(.initcall4s.init) *(.initcall5.init) *(.initcall5s.init) *(.initcallrootfs.init) *(.initcall6.init) *(.initcall6s.init) *(.initcall7.init) *(.initcall7s.init)`
3. `__initcall_end = .;`

这里 \_\_initcall\_start 指向 .initcall.init 节的开始， \_\_initcall\_end 指向它的结尾。而 .initcall.init 节又被分为几个子节。这个 subsys\_initcall 宏便是将指定的函数指针放在了 .initcall4.init 子节。

三：

\_\_attribute\_\_((packed)); 比如下边这个结构体(在 include/linux/usb/ch9.h 中定义)：

1. `struct usb_interface_descriptor {`
2. `__u8 bLength;`
3. `__u8 bDescriptorType;`
4. `__u8 bInterfaceNumber;`
5. `__u8 bAlternateSetting;`
6. `__u8 bNumEndpoints;`
7. `__u8 bInterfaceClass;`
8. `__u8 bInterfaceSubClass;`
9. `__u8 bInterfaceProtocol;`
10. `__u8 iInterface;`
11. `} __attribute__((packed));`

这里的 `__attribute__((packed))` 告诉编译器，这个结构的元素都是 1 字节对齐的，不要再添加填充位了。如果不给编译器这个暗示，编译器就会依据你的平台类型在结构的每个元素之间添加一定的填充位。

四：

1. `#define likely(x) __builtin_expect(!!(x),1)`
2. `#define unlikely(x) __builtin_expect(!!(x),0)`

`__builtin_expect` 是 GCC 里内建的一个函数：

1. `long __builtin_expect(long exp, long c)`

它的第一个参数 `exp` 为一个整型的表达式，返回值也是这个 `exp`，它的第二个参数 `c` 的值必须是一个编译器的常量，那这个内建函数的意思就是 `exp` 的预期值为 `c`，编译器可以根据这个信息适当的重排条件语句块的顺序，将符合这个条件的分支放在合适的地方。对于 `unlikely(x)` 就是告诉编译器 `x` 发生的可能性不大，那么这个条件块里语句的目标码可能就会被放到一个比较远的位置，以保证经常执行的目标码更紧凑，而 `likely` 相反。也就是说，如果你觉得 `if` 条件为 1 的可能性非常大时，可以在条件表达式外面包装一个 `likely()`，如果可能性非常小，则用 `unlikely()` 包装。

五：

1. `container_of(pointer,type,member);`

我们可以通过一个叫 `container_of` 的宏反查 `member` 所在的数据结构。比如：

1. `struct A_t{`
2. `int a;`
3. `int b;`
4. `};`
5. `struct A_t A;`
6. `int *b_p = &(A.b);`

使用如下方法可以从数据结构的一个元素出发，得到整个数据结构的指针。

1. `struct A_t *A_p = container_of(b_p, struct A_t, b);`

分享到：