[module\_init的内核调用顺序](http://blog.chinaunix.net/uid-27664726-id-4243961.html)

<http://blog.chinaunix.net/uid-27664726-id-4243961.html>

现在以module\_init为例分析initcall在内核中的调用顺序

在头文件init.h中，有如下定义：

#define module\_init(x)     \_\_initcall(x);

#define \_\_initcall(fn) device\_initcall(fn)

继续看：

#define device\_initcall(fn)              \_\_define\_initcall("6",fn,6)

#define \_\_define\_initcall(level,fn,id) \

       static initcall\_t \_\_initcall\_##fn##id \_\_used \

       \_\_attribute\_\_((\_\_section\_\_(".initcall" level ".init"))) = fn

最终我们看到的是module\_init的真身：\_\_define\_initcall(level,fn,id)，仔细推敲这个真身，知道这是个宏，它把传给module\_init的函数名组装成以\_\_initcall为前缀的、以6为后缀的函数名，并把这个函数定义到代码段.initcall6.init里面。

在代码段.initcall6.init里面？这函数躲在这里干嘛，啥时候才轮得到它出头啊！找到有此字符串的文件vmlinux.lds.h，相关代码如下所示：

#define INITCALLS                                           \

       \*(.initcallearly.init)                                        \

       VMLINUX\_SYMBOL(\_\_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)

在vmlinux.lds.S里面

\_\_initcall\_start = .;

                     INITCALLS

              \_\_initcall\_end = .;

              \_\_con\_initcall\_start = .;

                     \*(.con\_initcall.init)

              \_\_con\_initcall\_end = .;

              \_\_security\_initcall\_start = .;

                     \*(.security\_initcall.init)

              \_\_security\_initcall\_end = .;

言归正传，INITCALLS在\_\_initcall\_start = .与\_\_initcall\_end = .之间，表示INITCALLS宏内涵的相关段代码顺序存放在这里，module\_init所代表段的镶嵌其中

#define pure\_initcall(fn)          \_\_define\_initcall("0",fn,0)

#define core\_initcall(fn)          \_\_define\_initcall("1",fn,1)

#define core\_initcall\_sync(fn)        \_\_define\_initcall("1s",fn,1s)

#define postcore\_initcall(fn)           \_\_define\_initcall("2",fn,2)

#define postcore\_initcall\_sync(fn) \_\_define\_initcall("2s",fn,2s)

#define arch\_initcall(fn)          \_\_define\_initcall("3",fn,3)

#define arch\_initcall\_sync(fn)        \_\_define\_initcall("3s",fn,3s)

#define subsys\_initcall(fn)              \_\_define\_initcall("4",fn,4)

#define subsys\_initcall\_sync(fn)    \_\_define\_initcall("4s",fn,4s)

#define fs\_initcall(fn)                     \_\_define\_initcall("5",fn,5)

#define fs\_initcall\_sync(fn)            \_\_define\_initcall("5s",fn,5s)

#define rootfs\_initcall(fn)        \_\_define\_initcall("rootfs",fn,rootfs)

#define device\_initcall(fn)              \_\_define\_initcall("6",fn,6)

#define device\_initcall\_sync(fn)    \_\_define\_initcall("6s",fn,6s)

#define late\_initcall(fn)           \_\_define\_initcall("7",fn,7)

#define late\_initcall\_sync(fn)         \_\_define\_initcall("7s",fn,7s)

至此我们应该明白了各个initcall是如何来的以及在代码存储空间上是怎么组织的了吧，下面来看看内核是什么时候调用它的。

内核启动流程如下所示

Main()àlinux\_main()àstart\_uml()àstart\_kernel\_proc()àstart\_kernelàrest\_inità kernel\_inità do\_basic\_setupà do\_initcalls

do\_initcalls的代码如下所示

static void \_\_init do\_initcalls(void)

{

       initcall\_t \*call;

       for (call = \_\_early\_initcall\_end; call < \_\_initcall\_end; call++)

              do\_one\_initcall(\*call);

       /\* Make sure there is no pending stuff from the initcall sequence \*/

       flush\_scheduled\_work();

}

上面的代码中，\_\_early\_initcall\_end在INITCALLS内定义，\_\_initcall\_end在文件vmlinux.lds.S中定义，他们代表的是一些初始化函数的指针数组起始与结束地址，执行函数do\_initcalls时，包含在这各指针数组里面的函数顺序的被调用以执行一些必要的初始化工作。至此，明白的各initcall的执行时刻了吧。