你管这破玩意叫操作系统源码 | 第二回 自己给自己挪个地儿

Original 闪客 低并发编程 2021-11-14 16:30

收录于话题
#操作系统源码

26个

书接上回，上回书咱们说到，CPU 执行操作系统的最开始的两行代码。

```
mov ax,0x07c0
mov ds,ax
```

将数据段寄存器 ds 的值变成了 **0x07c0**，方便了之后访问内存时利用这个**段基址**进行寻址。

接下来我们带着这两行代码，继续往下看几行。

```
mov ax,0x07c0
mov ds,ax
mov ax,0x9000
mov es,ax
mov cx,#256
sub si,si
sub di,di
rep movw
```

此时 ds 寄存器的值已经是 0x07c0 了，然后又通过同样的方式将 **es** 寄存器的值变成 **0x9000**，接着又把 **cx** 寄存器的值变成 **256**（代码里确实是用十进制表示的，与其他地方有些不一致，不过无所谓）。

再往下看有两个 **sub** 指令，这个 sub 指令很简单，比如

```
sub a,b
```

就表示

```
a = a - b
```

那么代码中的

```
sub si,si
```

就表示

```
si = si - si
```

所以如果 sub 后面的两个寄存器一模一样，就相当于把这个寄存器里的值**清零**，这是一个基本玩法。

那就非常简单了，经过这些指令后，以下几个寄存器分别被附上了指定的值，我们梳理一下。

ds = 0x07c0
es = 0x9000
cx = 256
si = 0
di = 0

还记得上一讲画的 CPU 寄存器的总图么？此时就是这样了

CPU 中的关键寄存器

干嘛要给这些毫不相干的寄存器附上值呢？其实就是为下一条指令服务的，就是

```
rep movw
```

其中 **rep** 表示重复执行后面的指令。

而后面的指令 **movw** 表示复制一个**字**（word 16位），那其实就是**不断重复地复制一个字。**

那下面自然就有三连问：

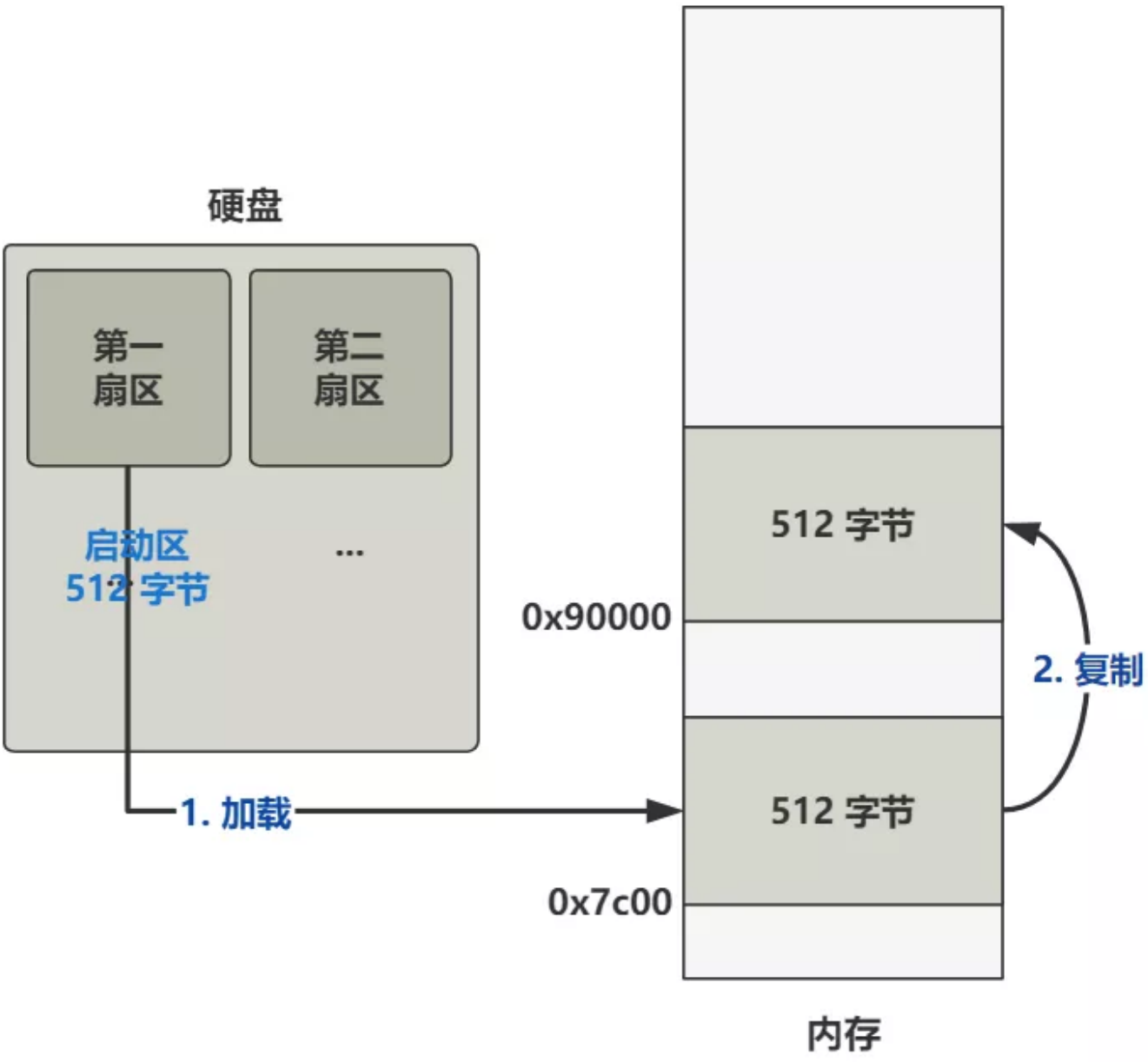
重复执行多少次呢？是 cx 寄存器中的值，也就是 256 次。

从哪复制到哪呢？是从 ds:si 处复制到 es:di 处。

一次复制多少呢？刚刚说过了，复制一个字，16 位，也就是两个字节。

上面是直译，那把这段话翻译成更人话的方式讲出来就是，**将内存地址 0x7c00 处开始往后的 512 字节的数据，原封不动复制到 0x90000 处。**

就是下图的第二步。



没错，就是这么折腾了一下。现在，操作系统最开头的代码，已经被挪到了 **0x90000** 这个位置了。

再往后是一个**跳转**指令。

```
jmpi go,0x9000
go:
  mov ax,cs
  mov ds,ax
```

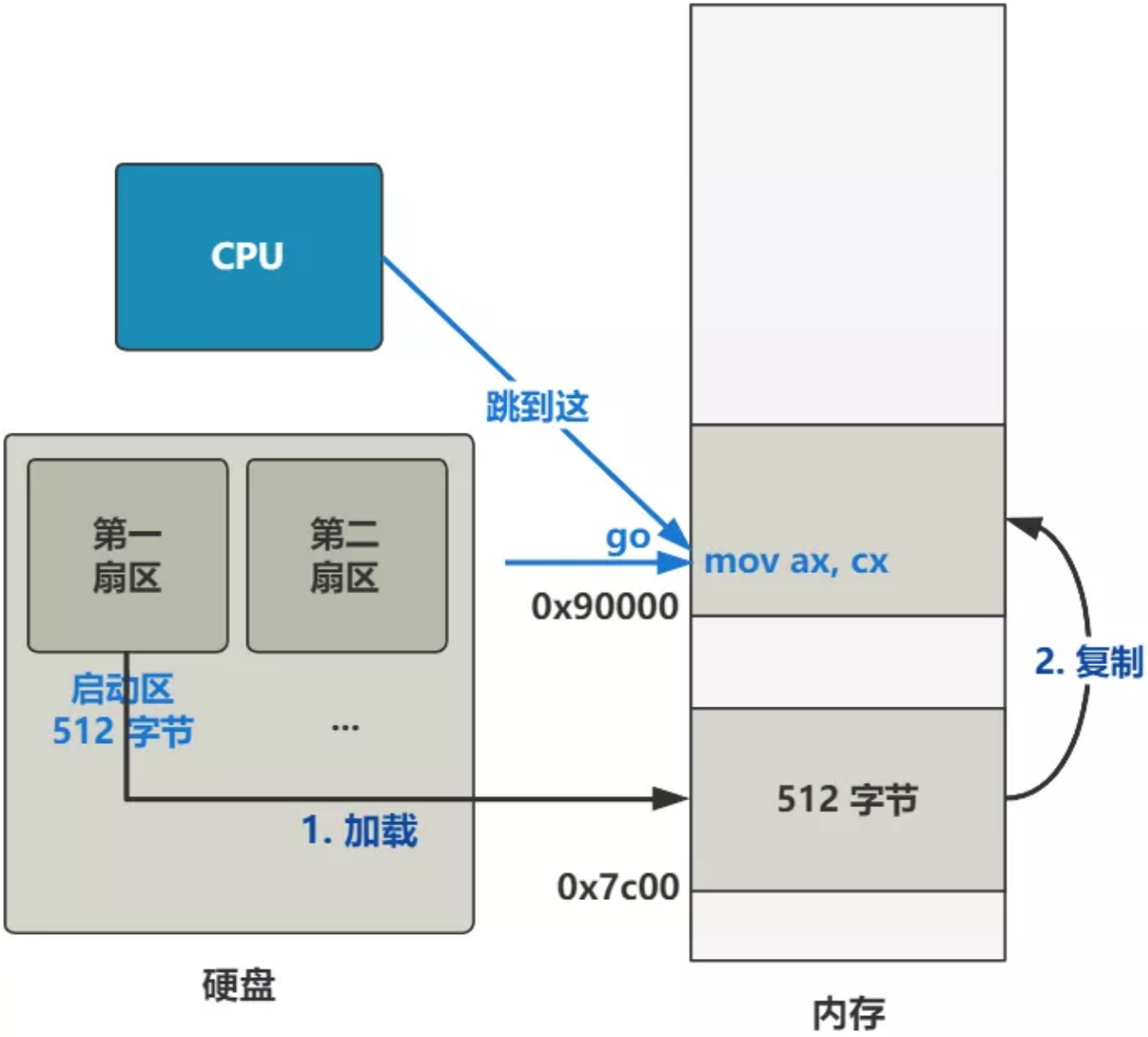
仔细想想或许你能猜到它想干嘛。

jmpi 是一个**段间跳转指令**，表示跳转到 **0x9000:go** 处执行。

还记得上一讲说的 **段基址：偏移地址** 这种格式的内存地址要如何计算吧？段基址仍然要先左移四位，因此结论就是跳转到 **0x90000 + go** 这个内存地址处执行。忘记的赶紧回去看看，这才过了一回哦，要稳扎稳打。

再说 go，go 就是一个**标签**，最终编译成机器码的时候会被翻译成一个值，这个值就是 go 这个标签在文件内的偏移地址。

这个偏移地址再加上 0x90000，就刚好是 go 标签后面那段代码 **mov ax,cs** 此时所在的内存地址了。



那假如 **mov ax,cx** 这行代码位于最终编译好后的二进制文件的 **0x08** 处，那 go 就等于 0x08，而最终 CPU 跳转到的地址就是 **0x90008** 处。

所以到此为止，前两回的内容，其实就是一段 **512** 字节的代码和数据，从硬盘的启动区先是被移动到了内存 **0x7c00** 处，然后又立刻被移动到 **0x90000** 处，并且跳转到此处往后再稍稍偏移 **go** 这个标签所代表的偏移地址处，也就是 **mov ax,cs** 这行指令的位置。

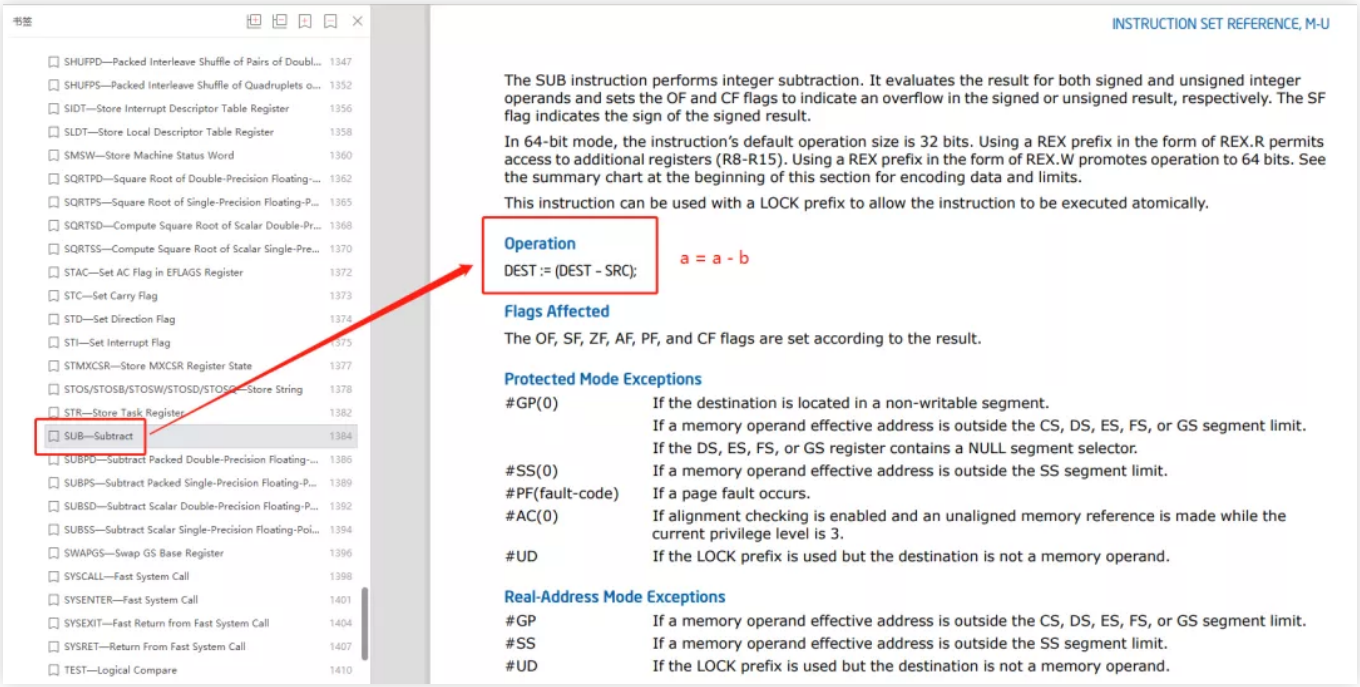
仍然是保持每回的简洁，本文就讲到这里，希望大家还跟得上，接下来的下一回，我们就把目光定位到 go 标签处往后的代码，看看他又要折腾些什么吧。

后面的世界越来越精彩，欲知后事如何，且听下回分解。

------- 本回扩展与延伸 -------

有关**寄存器**的详细信息，可以参考 Intel 手册：
Volume 1 Chapter 3.2 OVERVIEW OF THE BASIC EXECUTION ENVIRONMEN

如果想了解**汇编指令**的信息，可以参考 Intel 手册：
Volume 2 Chapter 3 ~ Chapter 5
比如本文出现的 sub 指令，你完全没必要去百度它的用法，直接看手册。



Intel 手册对于理解底层知识非常直接有效，但却没有很好的中文翻译版本，因此让许多人望而生畏，只能去看一些错误百出的中文二手资料和博客。因此我也发起了一个 **Intel 手册翻译计划**，就在阅读原文的 GitHub 里，感兴

趣的同胞们可以参与进来，我们共同完成一份伟大的事。

希望你跟完整个系列，收获的不仅仅是 Linux 0.11 源码的了解，更是自己探索问题和寻找答案的一个科学思考方式。

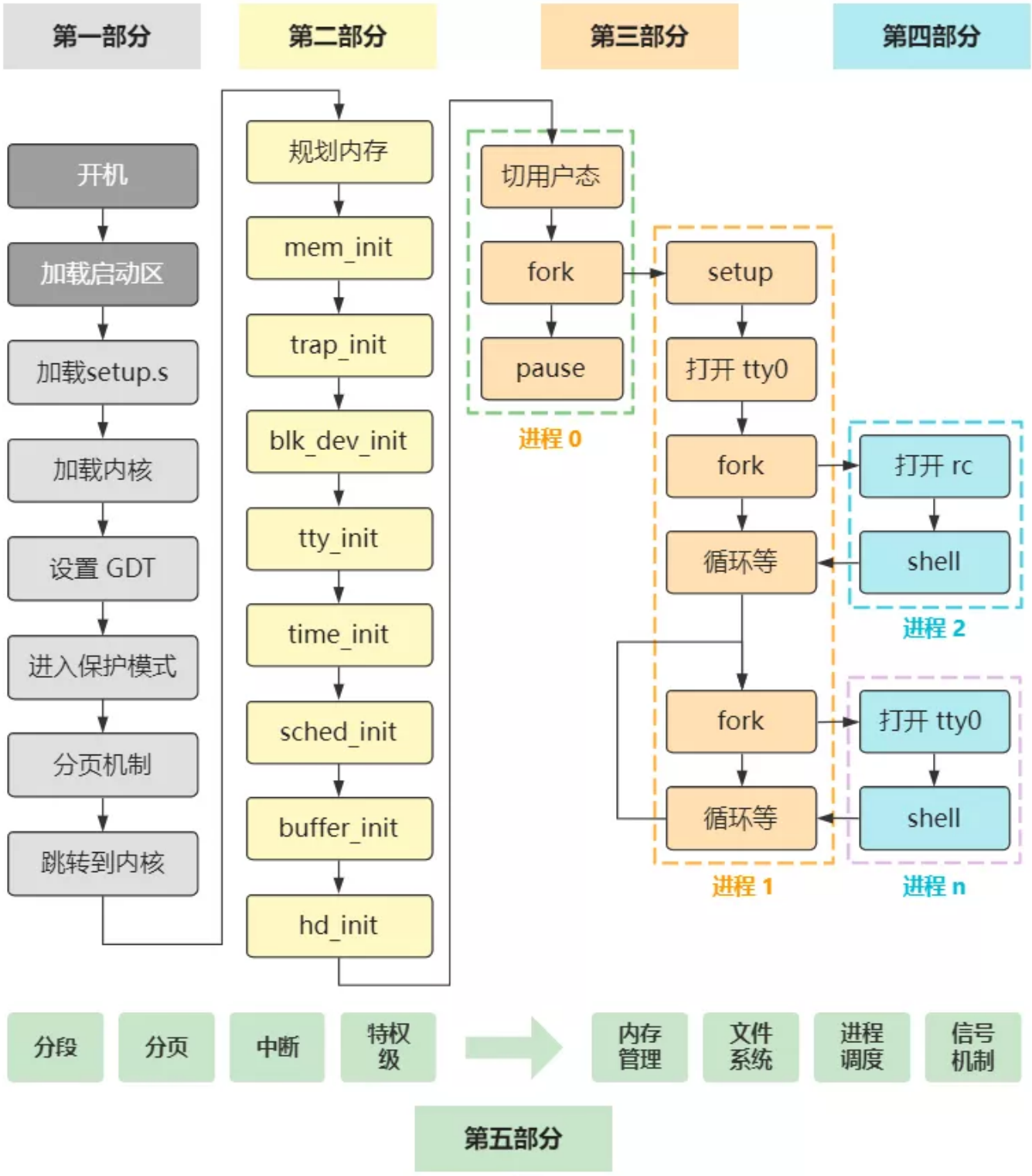
所以每次**本回扩展与延伸**这里，希望你也能每天进步一点点，实践起来，再不济，也能多学几个英语单词不是？

------- 关于本系列 -------

本系列的开篇词看这
闪客新系列！你管这破玩意叫操作系统源码

本系列的扩展资料看这（也可点击**阅读原文**），这里有很多有趣有价值的资料、答疑、互动参与项目，持续更新中，希望有你的参与。
https://github.com/sunym1993/flash-linux0.11-talk

本系列全局视角



最后，祝大家都能追更到系列结束，只要你敢持续追更，并且把每一回的内容搞懂，我就敢让你在系列结束后说一句，我对 Linux 0.11 很熟悉。

另外，本系列**完全免费**，希望大家能多多传播给同样喜欢的人，同时给我的 GitHub 项目点个 star，就在**阅读原文**处，这些就足够让我坚持写下去了！我们下回见。

低并发编程
战略上藐视技术，战术上重视技术
145篇原创内容

收录于话题 #操作系统源码 26

上一篇
你管这破玩意叫操作系统源码 | 第一回 最开始的两行代码

下一篇
你管这破玩意叫操作系统源码 | 第三回 做好最最基础的准备工作

Read more Modified on 2021/11/14