第20回 | 硬盘初始化 hd_init

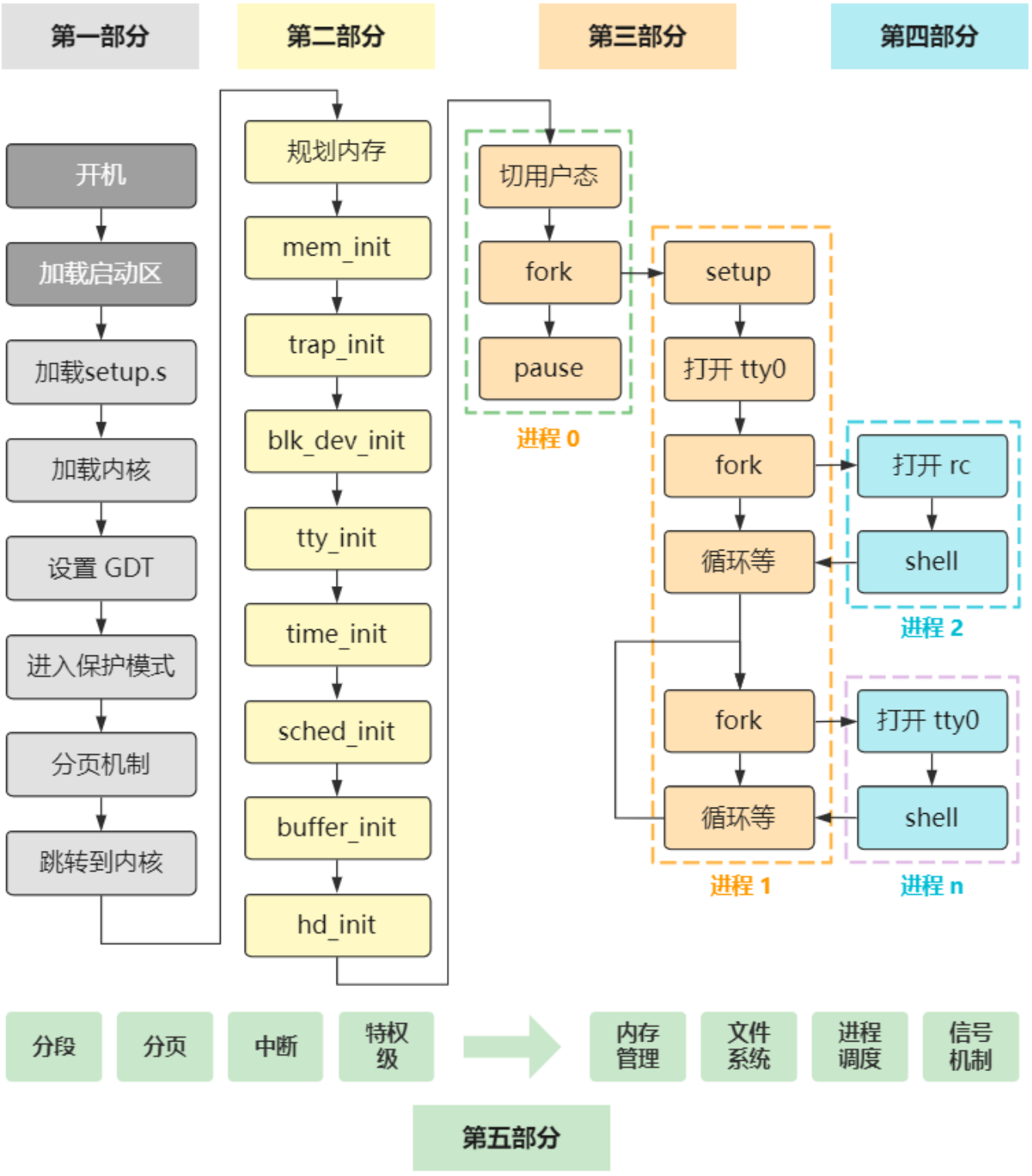
Original 闪客 低并发编程 2022-01-26 08:34

收录于合集
#操作系统源码

43个

新读者看这里，老读者直接跳过。

本系列会以一个读小说的心态，从开机启动后的代码执行顺序，带着大家阅读和赏析 Linux 0.11 全部核心代码，了解操作系统的技术细节和设计思想。



你会跟着我一起，看着一个操作系统从啥都没有开始，一步一步最终实现它复杂又精巧的设计，读完这个系列后希望你能发出感叹，原来操作系统源码就是这破玩意。

以下是**已发布文章**的列表，详细了解本系列可以先从开篇词看起。

开篇词

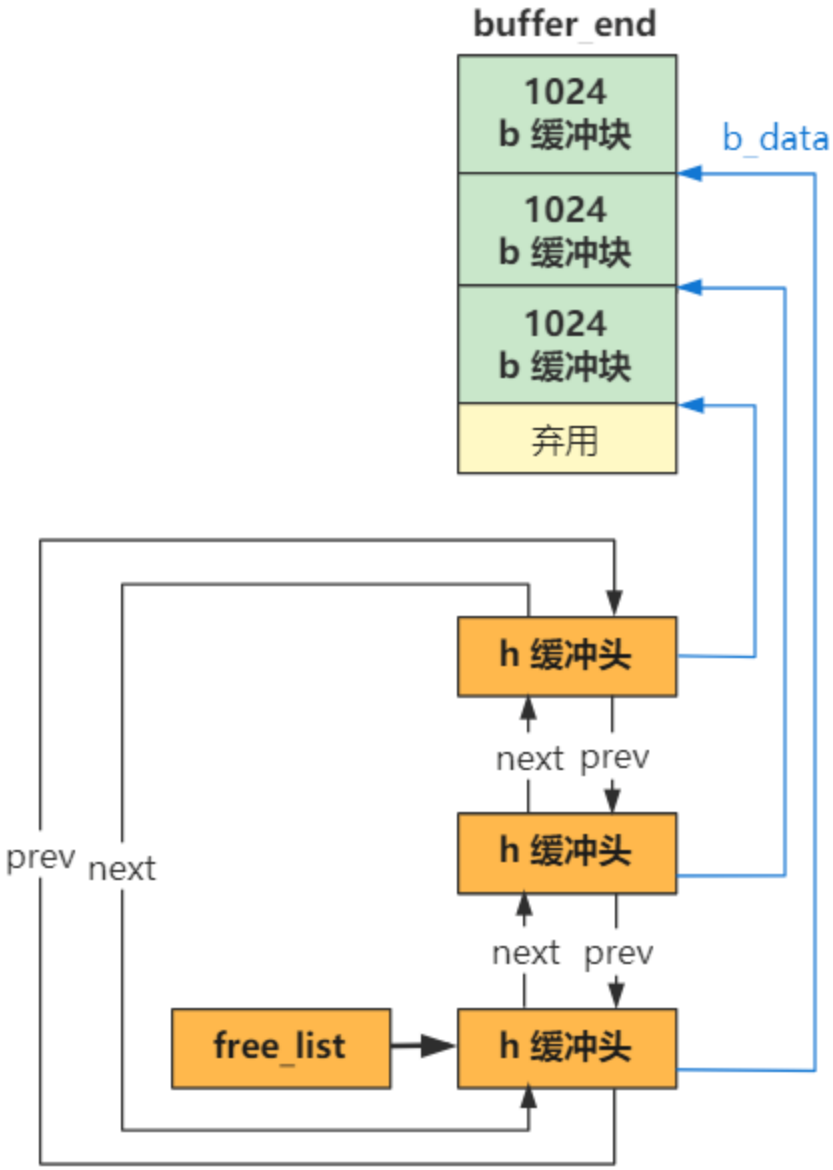
第一部分 进入内核前的苦力活

第二部分 大战前期的初始化工作

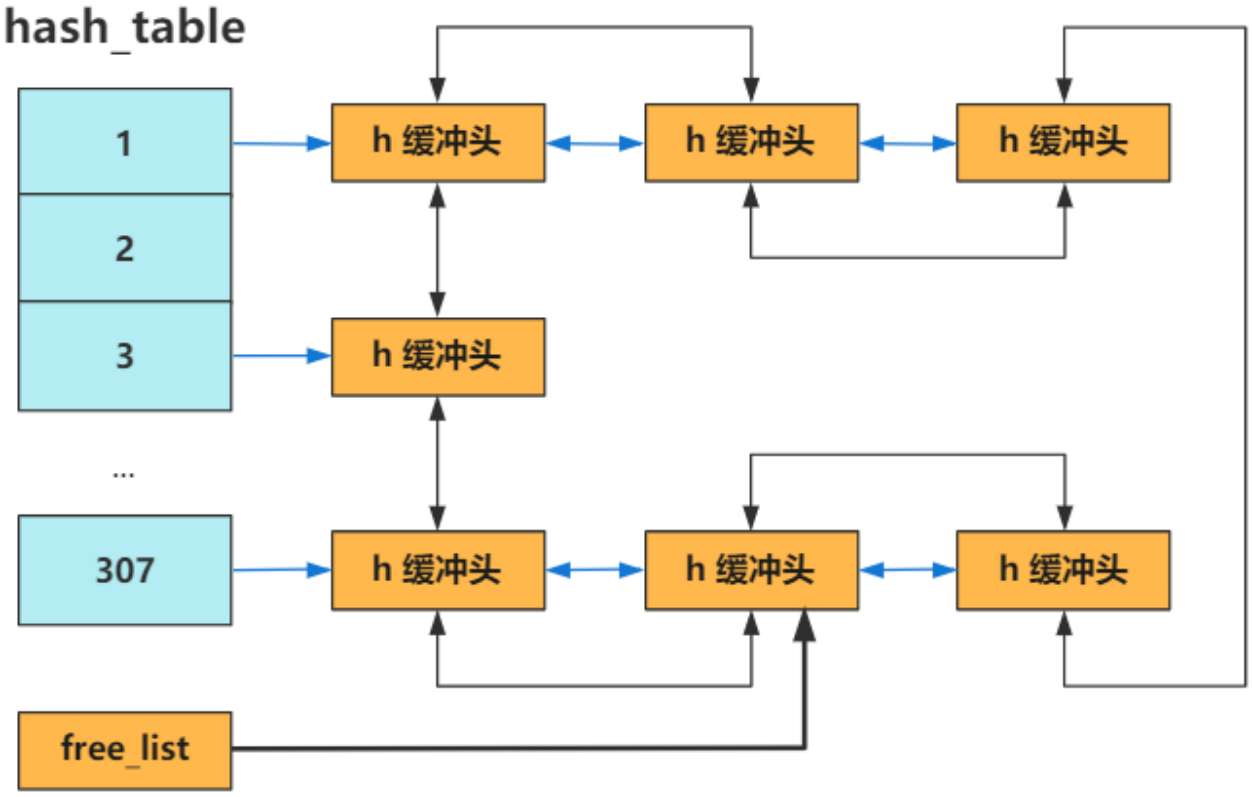
本系列的 GitHub 地址如下（文末阅读原文可直接跳转）
https://github.com/sunym1993/flash-linux0.11-talk

------- 正文开始 -------

书接上回，上回书咱们说到，`buffer_init` 完成了缓冲区初始化工作，内存中的缓冲区部分变成了这个样子。



而所有的缓冲头又被一个 **hash_table** 管理起来，以便查找。



至于缓冲区的这种管理怎么用，那等到后面文件系统中，如何读取一个块设备的数据，再展开讲解。

今天，我们看 main 函数中最后两个初始化函数！

```
void main(void) {
    ...
    mem_init(main_memory_start,memory_end);
    trap_init();
    blk_dev_init();
    chr_dev_init();
    tty_init();
    time_init();
    sched_init();
    buffer_init(buffer_memory_end);
    hd_init(); //本文重点
    floppy_init();

    sti();
    move_to_user_mode();
    if (!fork()) {init();}
    for(;;) pause();
}
```

最后两个了！兴不兴奋！

不过一口气看两个会不会消化不了？

不要担心，**hd_init** 是**硬盘初始化**，我们不得不看。

但 **floppy_init** 是**软盘初始化**，现在软盘几乎都被淘汰了，计算机中也没有软盘驱动器了，所以这个我们完全可以不看。

还记得小时候我特别喜欢收集软盘，里面分门别类存上我做的 Flash 动画，然后在软盘上的那个纸标签上写上文字，表示软盘存了什么，想想看还是回忆呢。

收，我们直接看 `hd_init` 这个硬盘初始化干了什么？

```
//struct blk_dev_struct {
//    void (*request_fn)(void);
//    struct request * current_request;
//};
//extern struct blk_dev_struct blk_dev[NR_BLK_DEV];

void hd_init(void) {
    blk_dev[3].request_fn = do_hd_request;
    set_intr_gate(0x2E,&hd_interrupt);
    outb_p(inb_p(0x21)&0xfb,0x21);
    outb(inb_p(0xA1)&0xbf,0xA1);
}
```

就这？一共就四行代码。

没错，初始化嘛，往往都比较简单，尤其是对硬件设备的初始化，大体都是：

1. 往某些 IO 端口上读写一些数据，表示开启它；

2. 然后再向中断向量表中添加一个中断，使得 CPU 能够响应这个硬件设备的动作；

3. 最后再初始化一些数据结构来管理。不过像是内存管理可能结构复杂些，外设的管理，相对就简单很多了。

看第一行代码：

```
void hd_init(void) {
    blk_dev[3].request_fn = do_hd_request;
    ...
}
```

我们把 **blk_dev** 数组索引 3 位置处的块设备管理结构 **blk_dev_struct** 的 **request_fn** 赋值为了 **do_hd_request**，这是啥意思呢？

因为有很多块设备，所以 Linux 0.11 内核用了一个 **blk_dev[]** 来进行管理，每一个索引表示一个块设备。

```
struct blk_dev_struct blk_dev[NR_BLK_DEV] = {
    { NULL, NULL },     /* no_dev */
    { NULL, NULL },     /* dev mem */
    { NULL, NULL },     /* dev fd */
    { NULL, NULL },     /* dev hd */
    { NULL, NULL },     /* dev ttyx */
    { NULL, NULL },     /* dev tty */
    { NULL, NULL }      /* dev lp */
};
```

你看，索引为 3 这个位置，就表示给硬盘 **hd** 这个块设备留的位置。

那么每个块设备执行读写请求都有自己的函数实现，在上层看来都是一个统一函数 **request_fn** 即可，具体实现各有不同，对于硬盘来说，这个实现就是 **do_hd_request** 函数。

是不是有点像接口？这其实就是**多态**思想在 C 语言的体现嘛~ 用 Java 程序员熟悉的话就是，父类引用 `request_fn` 指向子类对象 `do_hd_request` 的感觉咯。

我们再看第二行。

```
void hd_init(void) {
    ...
    set_intr_gate(0x2E,&hd_interrupt);
    ...
}
```

对于中断我们已经很熟悉了，这里就是又设置了一个新的中断，中断号是 **0x2E**，中断处理函数是 **hd_interrupt**，也就是说硬盘发生读写时，硬盘会发出中断信号给 CPU，之后 CPU 便会陷入中断处理程序，也就是执行 `hd_interrupt` 函数。

```
_hd_interrupt:
    ...
    xchgl _do_hd,%edx
    ...

// 如果是读盘操作，这个 do_hd 是 read_intr
static void read_intr(void) {
    ...
    do_hd_request();
    ...
}
```

好了，又多了一个中断，那我们再次梳理下目前开启的中断都有哪些。

中断号	中断处理函数
0 ~ 0x10	trap_init 里设置的一堆
0x20	timer_interrupt
0x21	keyboard_interrupt
0x2E	hd_interrupt
0x80	system_call

其中 **0-0x10** 这 17 个中断是 `trap_init` 里初始化设置的，是一些基本的中断，比如除零异常等。这个在 第14回 中断初始化 trap_init 有讲到。

之后，在控制台初始化 `con_init` 里，我们又设置了 **0x21** 键盘中断，这样按下键盘就有反应了。这个在 第16回 控制台初始化 tty_init 有讲到。

再之后，我们在进程调度初始化 `sched_init` 里又设置了 **0x20** 时钟中断，并且开启定时器。最后又偷偷设置了一个极为重要的 **0x80** 系统调用中断。这个在 第18回 进程调度初始化 sched_init 有讲到。

现在，我们在硬盘初始化 `hd_init` 里，又设置了硬盘中断，这样硬盘读写完成后将通过中断来通知 CPU。

上回书我提醒大家，有没有感觉到操作系统的中断驱动的特征，那本回再给大家展望一下。

> 看到最后，你会发现**操作系统就是一个靠中断驱动的死循环而已**，如果不发生任何中断，操作系统会一直在一个死循环里等待。换句话说，让操作系统工作的唯一方式，就是触发中断。

好了，再往下看后两行。

```
void hd_init(void) {
    ...
    outb_p(inb_p(0x21)&0xfb,0x21);
    outb(inb_p(0xA1)&0xbf,0xA1);
}
```

就是往几个 IO 端口上读写，其作用是**允许硬盘控制器发送中断请求信号**，仅此而已。我们向来是不深入硬件细节，知道往这个端口里写上这些数据，导致硬盘开启了中断，即可。

OK，本章就结束了，仅仅看初始化的工作，太简单了，连图都不用画就结束了。

当然 `hd.c` 里还有很多读写硬盘的方法，这个在之后文件系统用到他们时，自然会讲起，这里就抛个引子，看看读硬盘最最底层的操作流程，是怎样的。

我们看硬盘的端口表。

端口	读	写
0x1F0	数据寄存器	数据寄存器
0x1F1	错误寄存器	特征寄存器
0x1F2	扇区计数寄存器	扇区计数寄存器
0x1F3	扇区号寄存器或 LBA 块地址 0~7	扇区号或 LBA 块地址 0~7
0x1F4	磁道数低 8 位或 LBA 块地址 8~15	磁道数低 8 位或 LBA 块地址 8~15
0x1F5	磁道数高 8 位或 LBA 块地址 16~23	磁道数高 8 位或 LBA 块地址 16~23

端口	读	写
0x1F6	驱动器/磁头或 LBA 块地址 24~27	驱动器/磁头或 LBA 块地址 24~27
0x1F7	命令寄存器或状态寄存器	命令寄存器

那读硬盘就是，往除了第一个以外的后面几个端口写数据，告诉要读硬盘的哪个扇区，读多少。然后再从 `0x1F0` 端口一个字节一个字节的读数据。这就完成了一次硬盘读操作。

如果觉得不够具体，那来个具体的版本。

1. 在 `0x1F2` 写入要读取的扇区数

2. 在 `0x1F3 ~ 0x1F6` 这四个端口写入计算好的起始 LBA 地址

3. 在 `0x1F7` 处写入读命令的指令号

4. 不断检测 `0x1F7` （此时已成为状态寄存器的含义）的忙位

5. 如果第四步骤为不忙，则开始不断从 `0x1F0` 处读取数据到内存指定位置，直到读完

而操作系统的代码，也是这样写的，我们一睹为快一下，不用理解细节。

```
static void hd_out(unsigned int drive,unsigned int nsect,unsigned int sect,
        unsigned int head,unsigned int cyl,unsigned int cmd,
        void (*intr_addr)(void)) {
    ...
    do_hd = intr_addr;
    outb_p(hd_info[drive].ctl,HD_CMD);
    port = 0x1f0;
    outb_p(hd_info[drive].wpcom>>2,++port);
    outb_p(nsect,++port);
    outb_p(sect,++port);
    outb_p(cyl,++port);
    outb_p(cyl>>8,++port);
    outb_p(0xA0|(drive<<4)|head,++port);
    outb(cmd,++port);
}
```

看，那些 **outb_p** 方法，转换成汇编语言，就是 **out** 指令，往指定的硬盘 IO 端口上写数据，达到我们想要的读或者写的目的。

是不是很 low?

但我们由用户层写的各种 **read\write** 函数，即便是经过系统调用、文件系统、缓冲区管理等等过程，但只要是读写硬盘，最终都要调用到这个最底层的函数，殊途同归，逃不掉的！

好了，至此，我们就把所有的初始化工作，都讲完了！坚持读到现在的，都为自己鼓鼓掌！！！

下一回，我们将用整整一章的篇幅，完整梳理下我们所做的所有初始化工作，**这也标志着第二大部分正式完结**！这些初始化工作，将在后面的流程中，起到至关重要的作用，直到本系列结束。

欲知后事如何，且听下回分解。

如果觉得还行，给个 star 谢谢。

(点击阅读原文即可进入 Github 页)

------- 关于本系列 -------

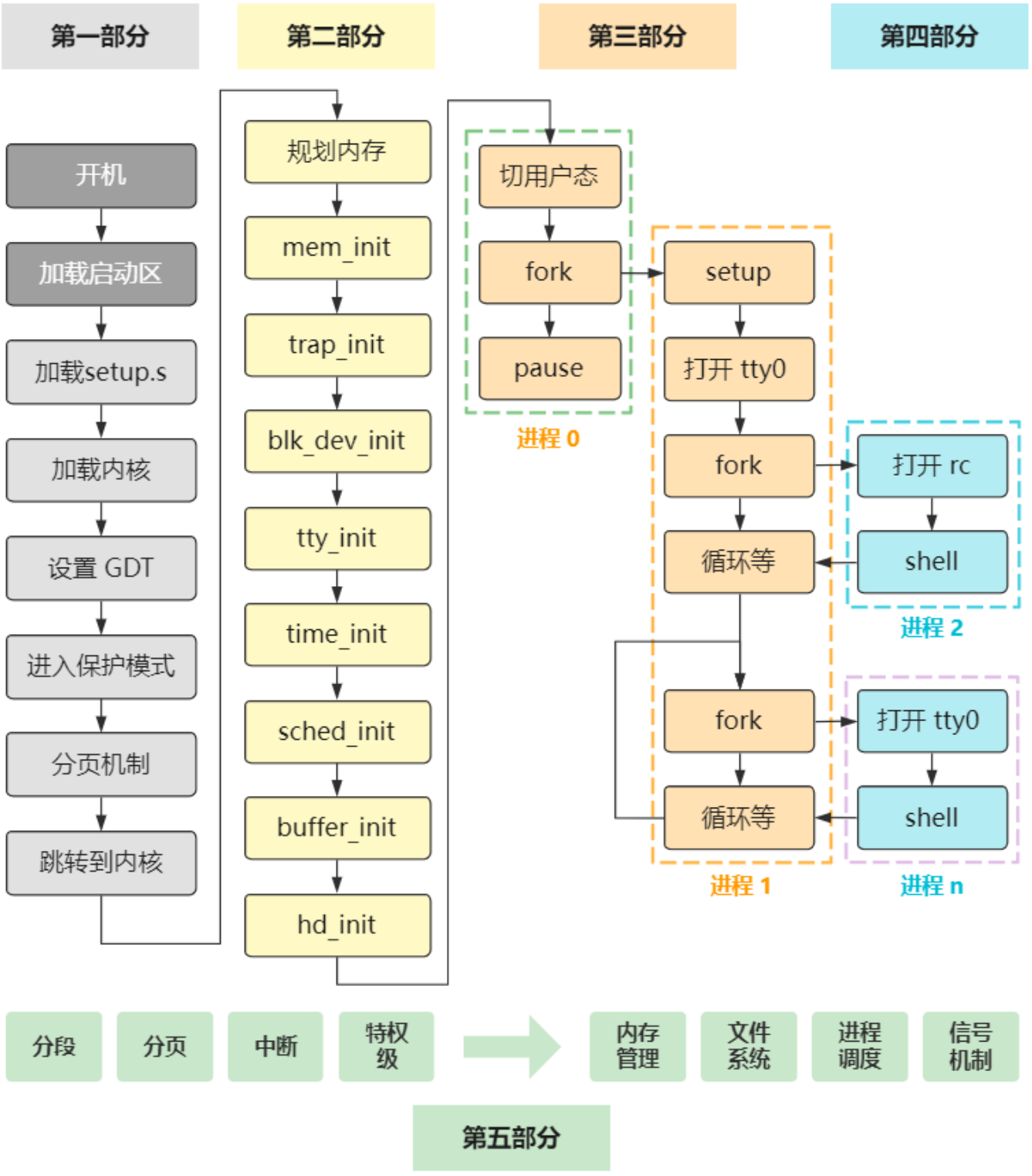
本系列的开篇词看这

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。

公众号更新系列文章不易，阅读量越来越低，希望大家多多传播，不方便的话点个小小的**赞**我也会很开心，谢谢大家咯。

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

175篇原创内容

Official Account

收录于合集 #操作系统源码 43

上一篇

第19回 | 操作系统就是用这两个面试常考的结构管理的缓冲区

下一篇

第二部分完结撒花！大战前期的初始化工作

Read more