第一部分完结 进入内核前的苦力活

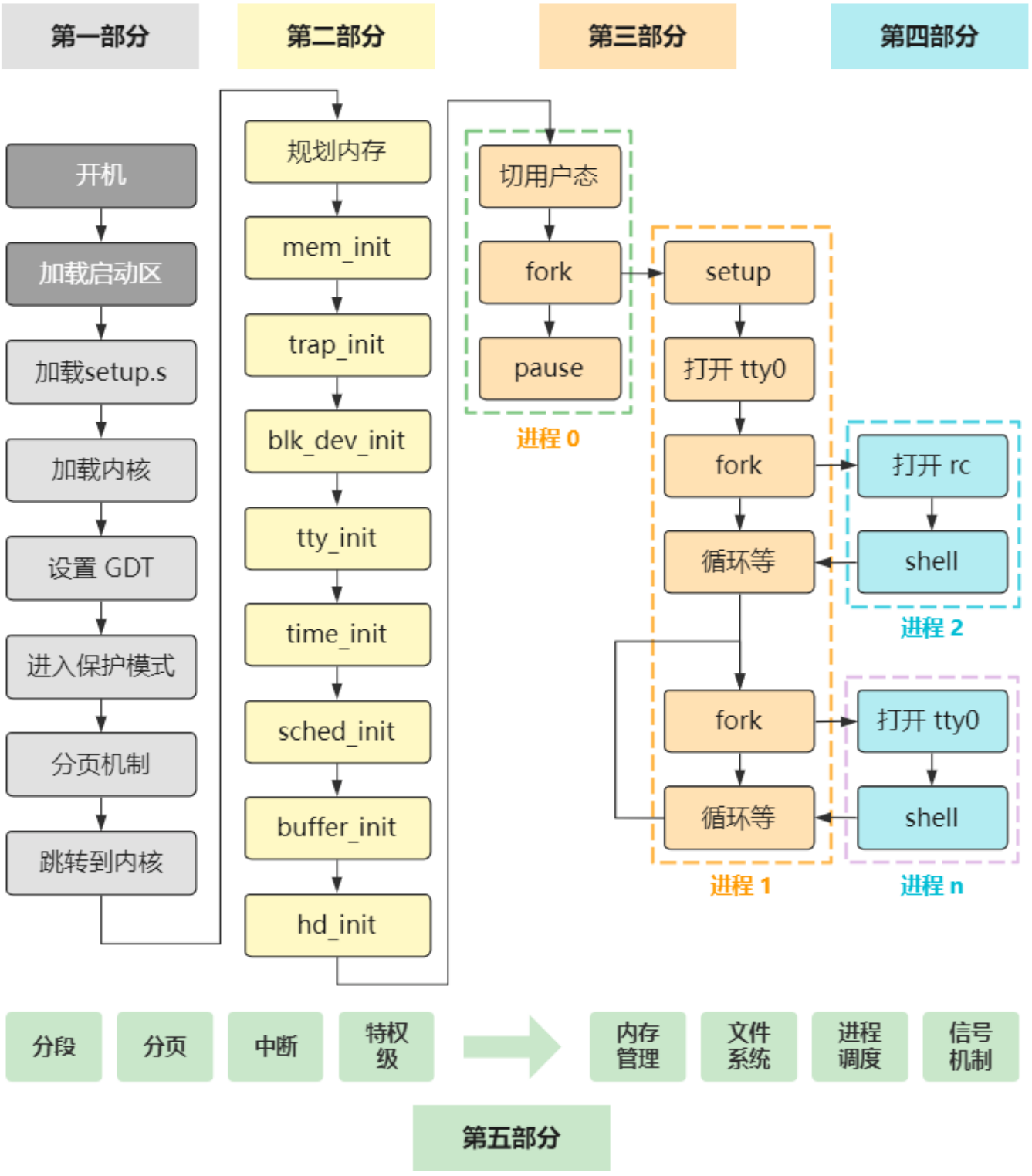
Original 闪客 低并发编程 2021-12-15 16:30

收录于合集
#操作系统源码

43个

第一部分总共有十回，这里是目录

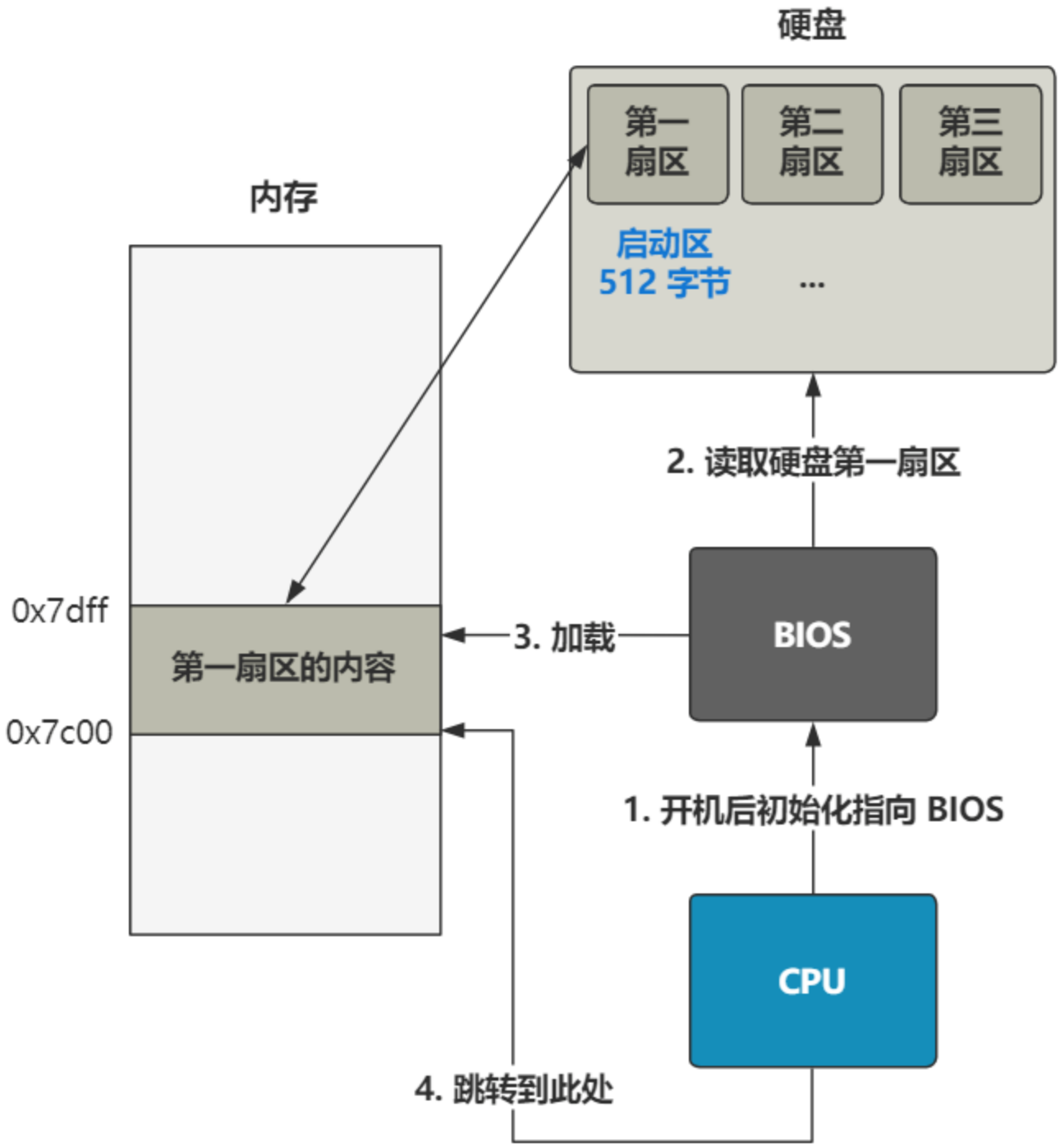
这张图展示了整个系列的结构



那今天就来整体梳理一下第一部分的内容，看过的同学跟着我回顾一下，没看过的同学，借着这波机会上车，也是不错的选择。

话不多说，我们开始。

当你按下开机键的那一刻，在主板上提前写死的固件程序 BIOS 会将硬盘中启动区的 512 字节的数据，原封不动复制到内存中的 0x7c00 这个位置，并跳转到那个位置进行执行，

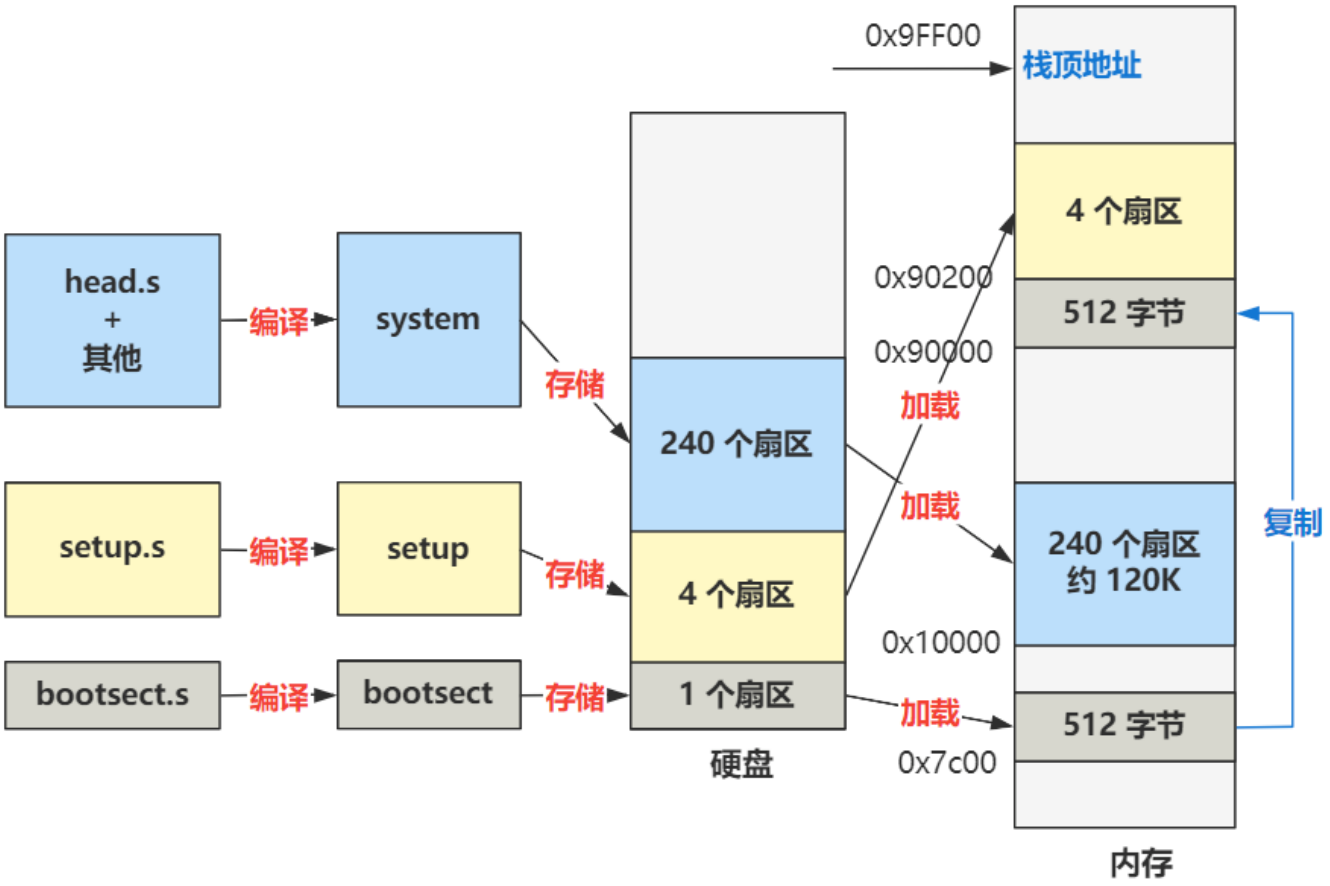


有了这个步骤之后，我们就可以把代码写在硬盘第一扇区，让 BIOS 帮我们加载到内存并由 CPU 去执行，我们不用操心这个过程。

而这一个扇区的代码，就是操作系统源码中最最最开始的部分，它可以执行一些指令，也可以把硬盘的其他部分加载到内存，其实本质上也是执行一些指令。这样，整个计算机今后如何运作，就完全交到我们自己的手中，想怎么玩就怎么玩了。

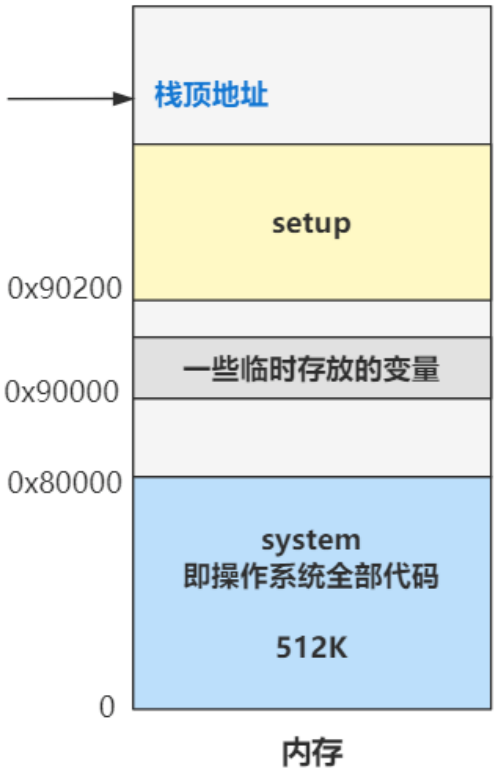
这是 第一回 | 最开始的两行代码 讲的内容。

接下来，直到 第四回 | 把自己在硬盘里的其他部分也放到内存来，我们才讲到整个操作系统的编译和加载过程的全貌，就是下面这张图。

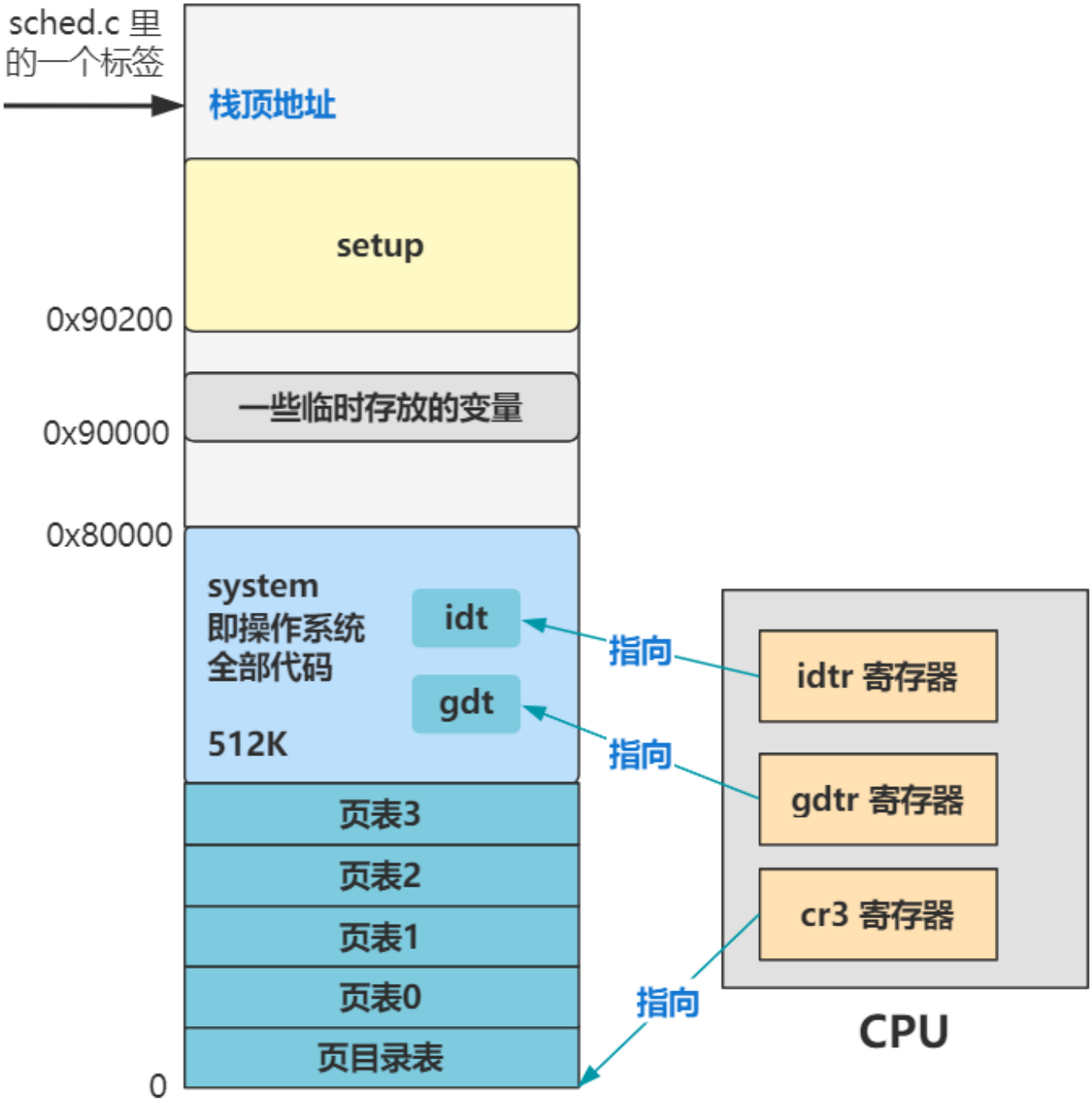


而我们整个的第一部分，其实就在讲 boot 文件夹下的三个汇编文件的内容，bootsect.s，setup.s 以及后面要和其他全部操作系统代码做链接的 head.s。

前五回的内容一直在调整内存的布局，把这块内存复制到那块，又把那块内存复制到这块，所以在 第五回 | 进入保护模式前的最后一次折腾内存 的结尾，我让你记住这样一张图，在很长一段时间这个内存布局的大体框架就不会再变了，前五回的内容你也可以抛在脑后了。

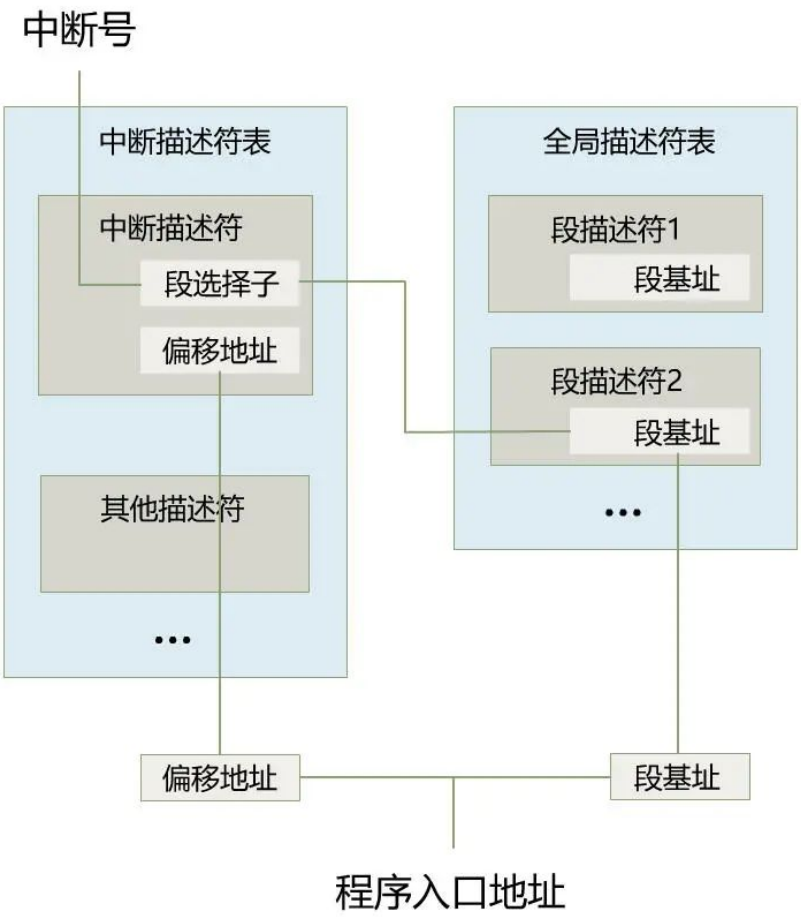


从第六回开始往后，就是逐渐进入保护模式，并设置分段、分页、中断等机制的地方。最终的内存布局变成了这个样子。

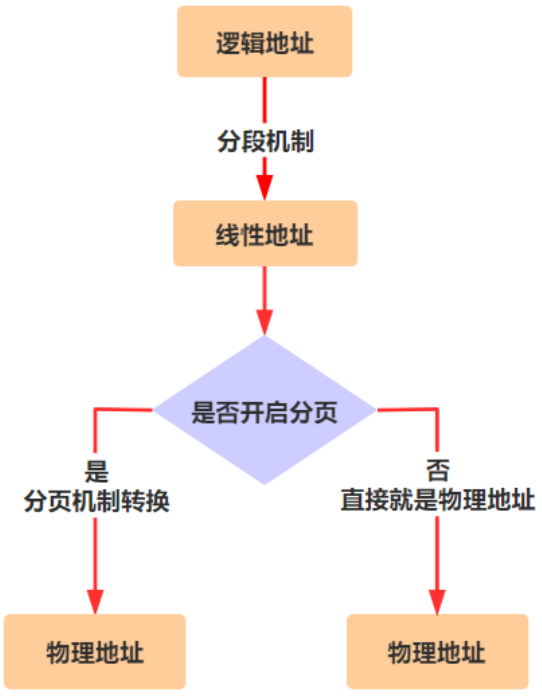


你看，idtr 寄存器指向了 idt，这个就是中断的设置；gdtr 寄存器指向了 gdt，这个就是全局描述符表的设置，可以简单理解为分段机制的设置；cr3 寄存器指向了页目录表的位置，这个就是分页机制的设置。

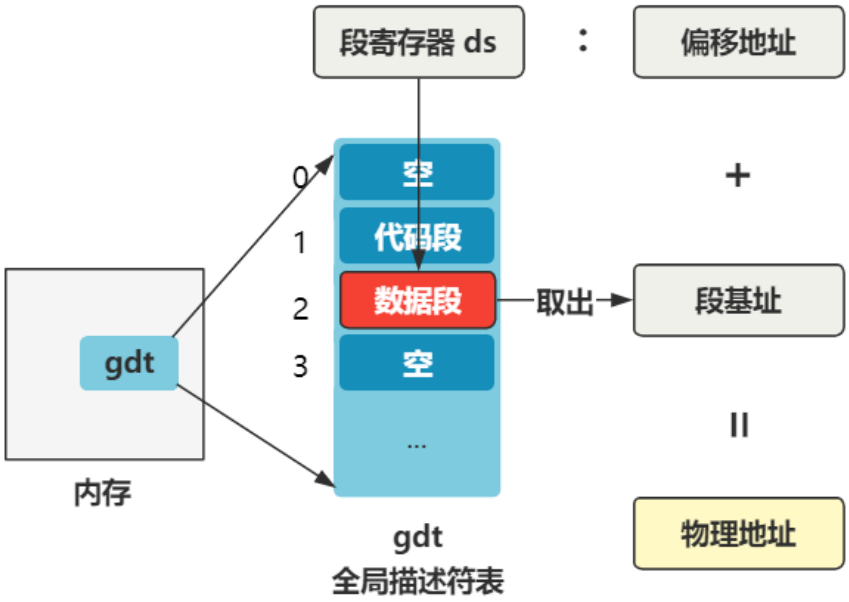
中断的设置，就引出了 CPU 与操作系统处理中断的流程。



分段和分页的设置，引出了逻辑地址到物理地址的转换。

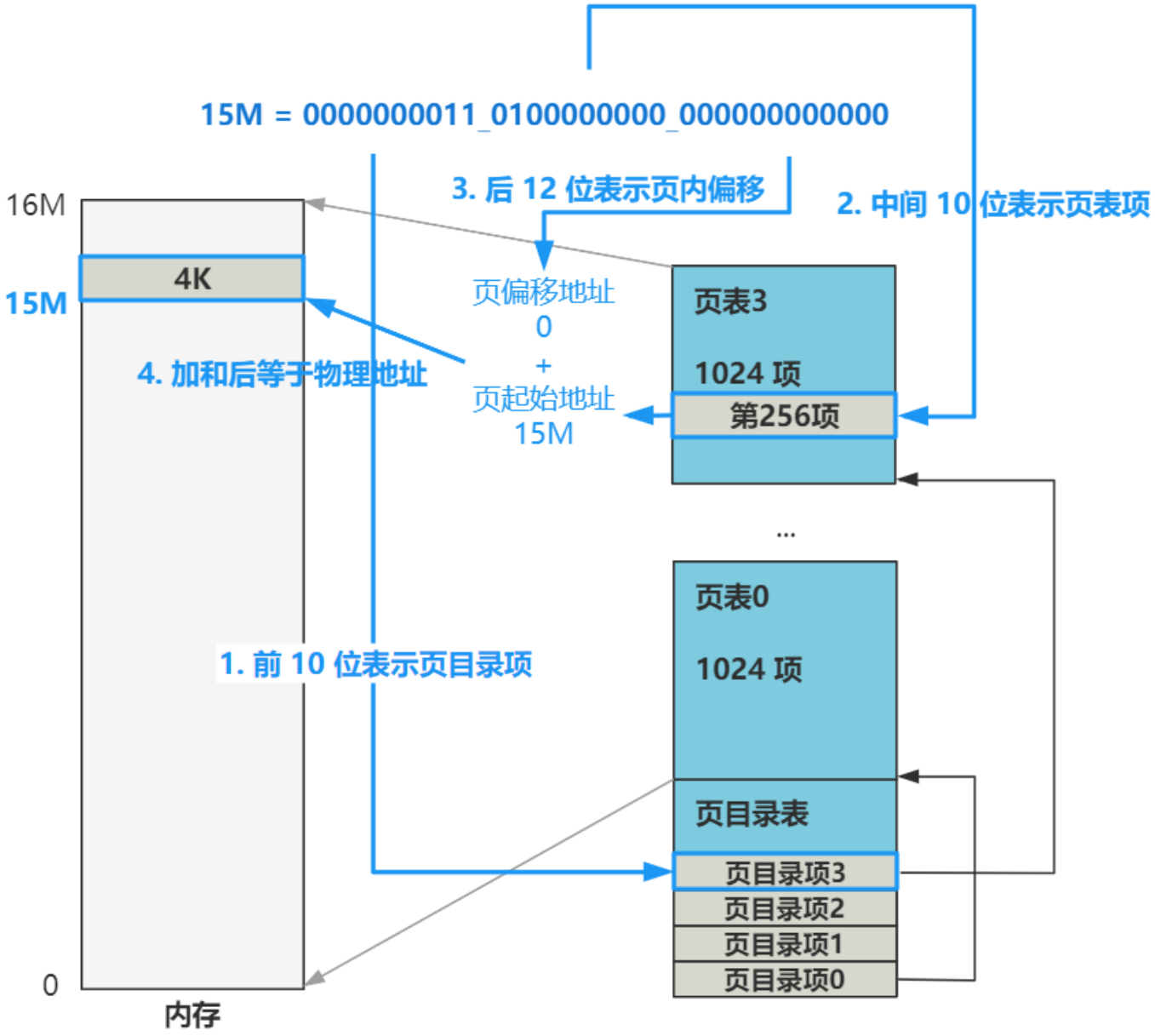


具体来说，逻辑地址到线性地址的转换，依赖 Intel 的分段机制。



保护模式下物理地址的转换（仅段机制）

而线性地址到物理地址的转换，依赖 Intel 的分页机制。



分段和**分页**，就是 Intel 管理内存的两大利器，也是内存管理最最最最底层的支撑。

而 Intel 本身对于访问内存就分成三类：

代码
数据
栈

而 Intel 也提供了三个段寄存器来分别对应着三类内存：

代码段寄存器（cs）
数据段寄存器（ds）
栈段寄存器（ss）

具体来说：

cs:eip 表示了我们要执行哪里的代码。
ds:xxx 表示了我们要访问哪里的数据。
ss:esp 表示了我们的栈顶地址在哪里。

而第一部分的代码，也做了如下工作：

将 **ds** 设置为了 0x10，表示指向了索引值为 2 的全局描述符，即数据段描述符。
将 **cs** 通过一次长跳转指令设置为了 8，表示指向了索引值为 1 的全局描述符，即代码段描述符。
将 **ss:esp** 这个栈顶地址设置为 user_stack 数组的末端。

你看，分段和分页，以及这几个寄存器的设置，其实本质上就是安排我们今后访问内存的方式，做了一个初步规划，**包括去哪找代码、去哪找数据、去哪找栈，以及如何通过分段和分页机制将逻辑地址转换为最终的物理地址。**

而所有上面说的这一切，和 Intel CPU 这个硬件打交道比较多，设置了一些最最最最基础的环境和内存布局，为之后进入 main 函数做了充分的准备，因为 c 语言虽然很底层了，但也有其不擅长的事情，就交给第一部分的汇编语言来做，所以我称第一部分为**进入内核前的苦力活。**

接下来，也就是从第二部分开始，我将会讲述 main.c 里的 main 函数，短短几行，包含了操作系统的全部核心思想。

```
void main(void) {
    ROOT_DEV = ORIG_ROOT_DEV;
    drive_info = DRIVE_INFO;
    memory_end = (1<<20) + (EXT_MEM_K<<10);
    memory_end &= 0xfffff000;
    if (memory_end > 16*1024*1024)
        memory_end = 16*1024*1024;
    if (memory_end > 12*1024*1024)
        buffer_memory_end = 4*1024*1024;
    else if (memory_end > 6*1024*1024)
        buffer_memory_end = 2*1024*1024;
    else
        buffer_memory_end = 1*1024*1024;
    main_memory_start = buffer_memory_end;
    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();
    floppy_init();
    sti();
    move_to_user_mode();
    if (!fork()) {
        init();
    }
    for(;;) pause();
}
```

敬请期待吧!

另外，前十回几乎每一回都有资料扩展部分，基本是围绕着 Intel 手册，把一些文中提到的知识点在一手资料中给出标准答案，大家可以多看看，培养下自己看一手资料的习惯。

由此也可以看出，前十回的苦力活，大部分是在和 Intel CPU 这个硬件打交道，因此阅读 Intel 技术手册从而了解 CPU 体系结构和机制，是理解这一切最直接和有效的办法。

以下列出我所有让大家扩展阅读的资料

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
如果想了解计算机启动时详细的初始化过程，还是得参考 Intel 手册：
Volume 3A Chapter 9 PROCESSOR MANAGEMENT AND INITIALIZATION
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
保护模式下逻辑地址到线性地址（不开启分页时就是物理地址）的转化，看 Intel 手册：
Volume 3 Chapter 3.4 Logical And Linear Addresses
关于逻辑地址-线性地址-物理地址的转换，可以参考 Intel 手册：
Intel 3A Chapter 3 Protected-Mode Memory Management
段描述符结构和详细说明，看 Intel 手册：
Volume 3 Chapter 3.4.5 Segment Descriptors
页目录表和页表的具体结构，可以看
Intel 3A Chapter 4.3 32-bit paging
关于 ret 指令，其实 Intel CPU 是配合 call 设计的，有关 call 和 ret 指令，即调用和返回指令，可以参考 Intel 手册：
Intel 1 Chapter 6.4 CALLING PROCEDURES USING CALL AND RET

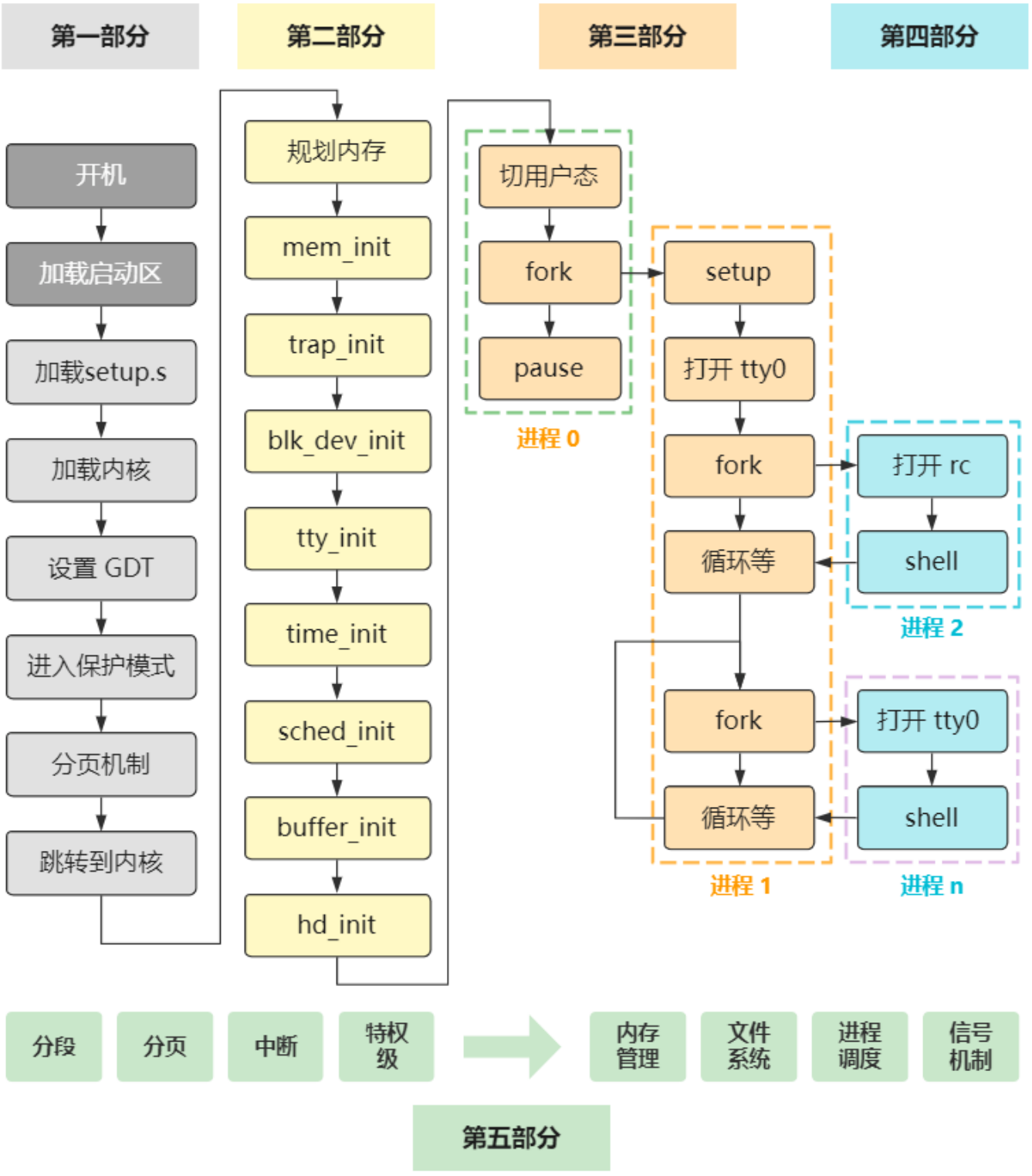
资料就摆在你眼前了，你再不去看，我就没办法咯，加油！

------- 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。

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
第十回 | 进入 main 函数前的最后一跃！

下一篇
第11回 | 整个操作系统就 20 几行代码

Read more

People who liked this content also liked

我常用的5个效率小工具，强烈推荐
我们谈论数据科学

C++11新特性之decltype类型推导
三贝勒文子

不足为奇的过往
小陈Teacher