第38回 | 操作系统启动完毕！

Original 闪客 低并发编程 2022-05-25 17:30 Posted on 北京

收录于合集
#操作系统源码

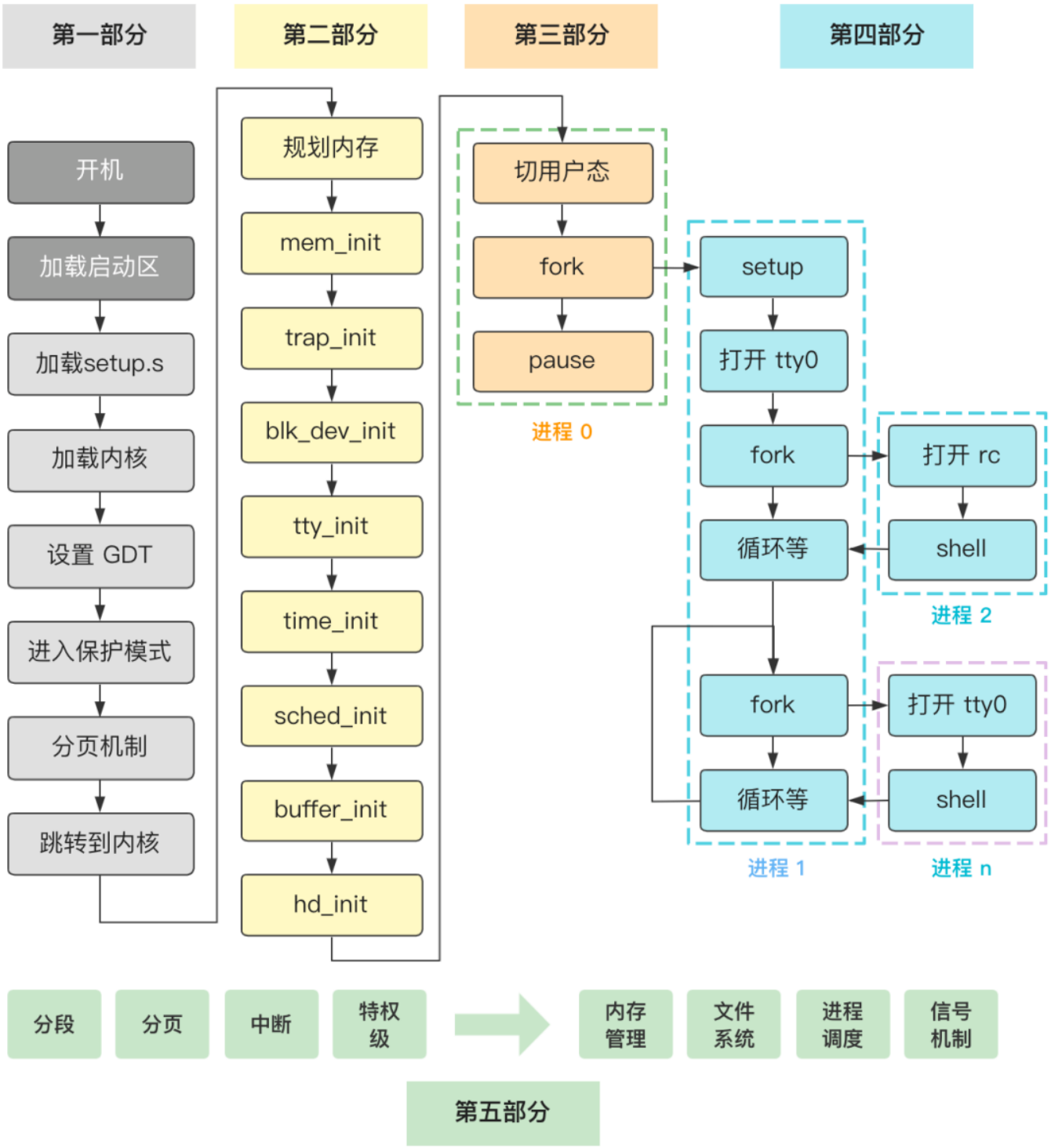
43个

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。

本系列的 GitHub 地址如下，希望给个 star 以示鼓励（文末**阅读原文**可直接跳转，也可以将下面的链接复制到浏览器里打开）
https://github.com/sunym1993/flash-linux0.11-talk

本回的内容属于第四部分。



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

第三部分：一个新进程的诞生

第四部分：shell 程序的到来

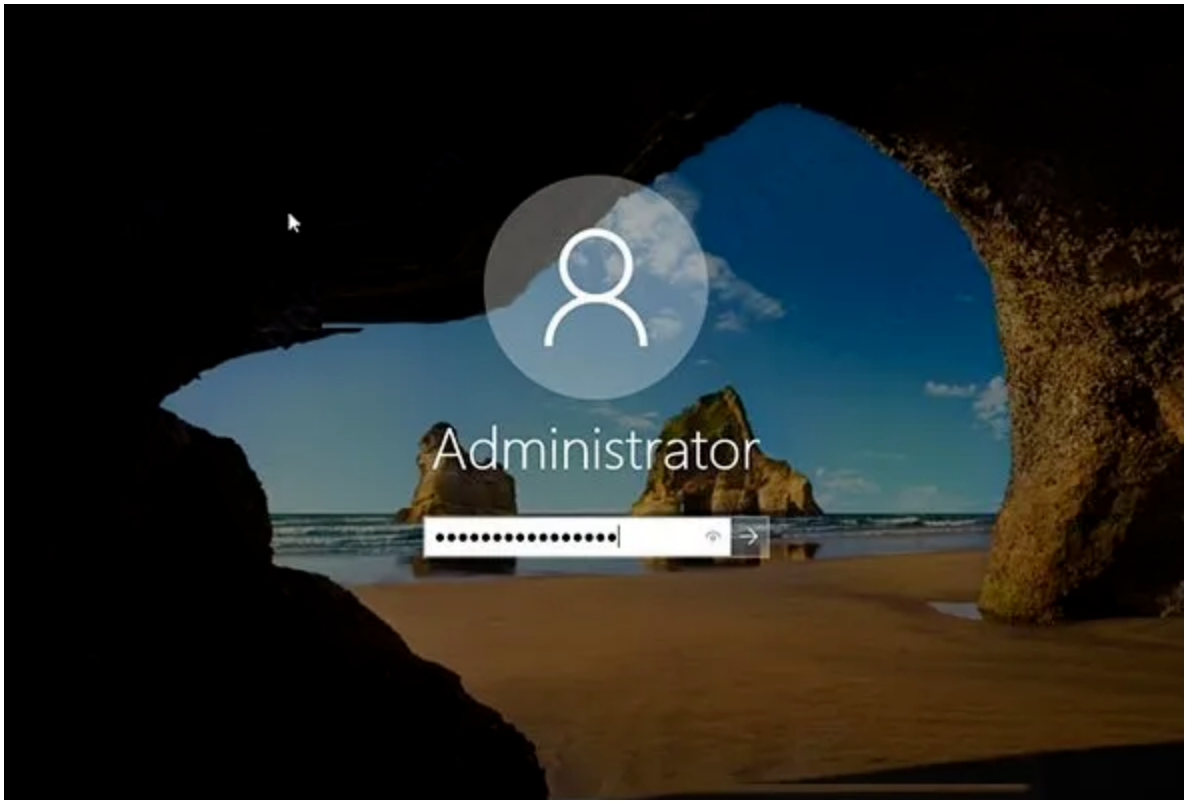
------- 正文开始 -------

书接上回，上回书咱们说到一个 shell 程序的执行原理，至此我们的操作系统终于将控制权转交给了 shell，由 shell 程序和我们人类进行友好的交互。

其实到这里，操作系统的使命就基本结束了。

此时我想到了之前有人问过我的一个问题，他说为什么现在的电脑开机后和操作系统启动前，还隔着好长一段时间，这段时间运行的代码是什么？

在我的继续追问下才知道，他说的操作系统的开始部分，是我们看到了诸如 Windows 登陆画面的时候。



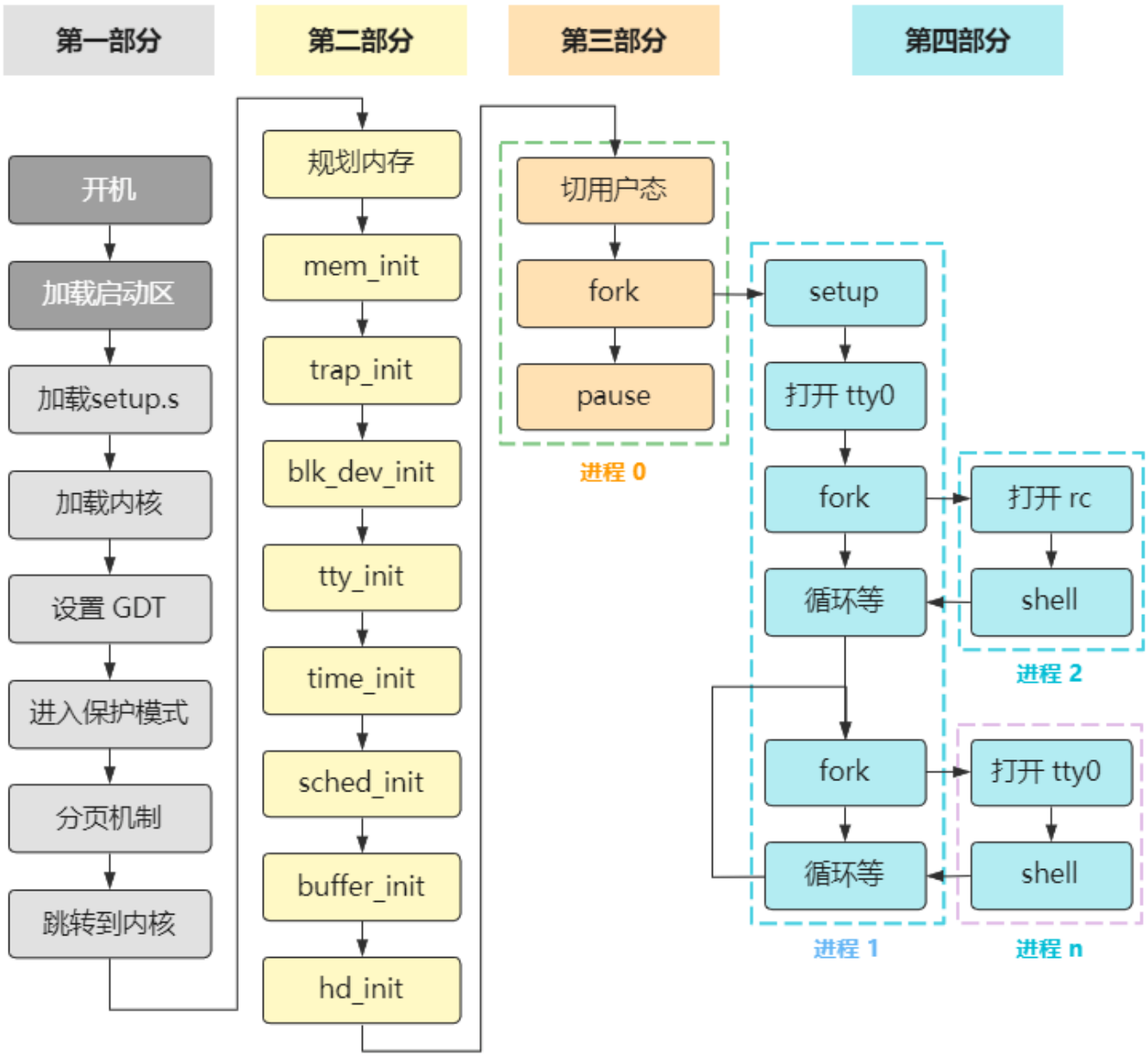
这个登陆画面就和我们 Linux 0.11 里讲的这个 shell 程序一样，已经可以说标志着操作系统启动完毕了，通过 shell 不断接受用户命令并执行命令的死循环过程中。

甚至在 Linux 0.11 里根本都找不到 shell 的源代码，说明 Linux 0.11 并没有认为 shell 是操作系统的一部分，它只是个普通的用户程序，和你在操作系统里自己写个 hello world 编译成 a.out 执行一样。在执行这个 shell 程序前已经可以认为操作系统启动完毕了。

操作系统就是初始化了一堆数据结构进行管理，并且提供了一揽子**系统调用**接口供上层的应用程序调用，仅此而已。再多做点事就是提供一些常用的用户程序，但这不是必须的。

OK，上一回我留了一个问题，shell 程序执行了，操作系统就结束了么？

此时我们不妨从宏观视角来看一下当前的进度。



看最右边的蓝色部分的流程即可。

我们先是建立了操作系统的一些最基本的环境与管理结构，然后由进 0 fork 出处于用户态执行的进程 1，进程 1 加载了文件系统并打开终端文件，紧接着就 fork 出了进程 2，进程 2 通过我们刚刚讲述的 execve 函数将自己替换成了 shell 程序。

如果看代码的话，其实我们此时处于一个以 rc 为标准输入的 shell 程序。

```
// main.c
void main(void) {
    ...
    if (!fork()) {
        init();
    }
    for(;;) pause();
}

void init(void) {
    ...
    // 一个以 rc 为标准输入的 shell
    if (!(pid=fork())) {
        ...
        open("/etc/rc",O_RDONLY,0);
        execve("/bin/sh",argv_rc,envp_rc);
    }
    // 等待这个 shell 结束
    if (pid>0)
        while (pid != wait(&i))
    ...
    // 大的死循环，不再退出了
    while (1) {
        // 一个以 tty0 终端为标准输入的 shell
        if (!(pid=fork())) {
            ...
            (void) open("/dev/tty0",O_RDWR,0);
            execve("/bin/sh",argv,envp);
        }
        // 这个 shell 退出了继续进大的死循环
        while (1)
            if (pid == wait(&i))
                break;
        ...
    }
}
```

就是 open 了 /etc/rc 然后 execve 了 /bin/sh 的这个程序，代码中标记为蓝色的部分。

shell 程序有个特点，就是如果标准输入为一个普通文件，比如 /etc/rc，那么文件读取后就会使得 shell 进程退出，如果是字符设备文件，比如由我们键盘输入的 /dev/tty0，则不会使 shell 进程退出。

这就使得标准输入为 /etc/rc 文件的 shell 进程在读取完 /etc/rc 这个文件并执行这个文件里的命令后，就退出了。

所以，这个 /etc/rc 文件可以写一些你觉得在正式启动大死循环的 shell 程序之前，要做的一些事，比如启动一个登陆程序，让用户输入用户名和密码。

好了，那作为这个 shell 程序的父进程，也就是进程 0，在检测到 shell 进程退出后，就会继续往下走。

```
// main.c
void init(void) {
    ...
    // 一个以 rc 为标准输入的 shell
    ...
    // 等待这个 shell 结束
    if (pid>0)
        while (pid != wait(&i))
    ...
    // 大的死循环，不再退出了
    while (1) {
        ...
    }
}
```

下面的 **while(1)** 死循环里，是和创建第一个 shell 进程的代码几乎一样。

```
// main.c
void init(void) {
    ...
    // 大的死循环，不再退出了
    while (1) {
        // 一个以 tty0 终端为标准输入的 shell
        if (!(pid=fork())) {
            ...
            (void) open("/dev/tty0",O_RDWR,0);
            execve("/bin/sh",argv,envp);
        }
        // 这个 shell 退出了继续进大的死循环
        while (1)
            if (pid == wait(&i))
                break;
        ...
    }
}
```

只不过它的标准输入被替换成了 **tty0**，也就是接受我们键盘的输入。

这个 shell 程序不会退出，它会不断接受我们键盘输入的命令，然后通过 fork+execve 函数执行我们的命令，这在上一回讲过了。

当然，如果这个 shell 进程也退出了，那么操作系统也不会跳出这个大循环，而是继续重试。

整个操作系统到此为止，看起来就是这个样子。

```
// main.c
void main() {
    // 初始化环境
    ...
    // 外层操作系统大循环
    while(1) {
        // 内层 shell 程序小循环
        while(1) {
            // 读取命令 read
            ...
            // 创建进程 fork
            ...
            // 执行命令 execve
            ...
        }
    }
}
```

当然，这只是表层的。

除此之外，这里所有的键盘输入、系统调用、进程调度，统统都需要**中断**来驱动，所以很久之前我说过，**操作系统就是个中断驱动的死循环**，就是这个道理。

OK！到此为止，操作系统终于启动完毕，达到了怠速的状态，它本身设置好了一堆中断处理程序，随时等待着中断的到来进行处理，同时它运行了一个 shell 程序用来接受我们普通用户的命令，以同人类友好的方式进行交互。

完美！

欲知后事如何，且听下回分解。

------- 关于本系列 -------

本系列的开篇词看这，开篇词

本系列的番外故事看这，让我们一起来写本书？也可以直接无脑加入星球，共同参与这场旅行。

最后，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37回 | shell 程序跑起来了

下一篇
第四部分完结！操作系统启动完毕！

Read more

People who liked this content also liked

今天我下了个JDK

低并发编程

第四部分完结！操作系统启动完毕！

低并发编程

HeyUI-Admin（前端开发者们的福利来了）

RunTheCode