第34回 | 进程2的创建

Original 闪客 低并发编程 2022-04-20 17:30

收录于合集
#操作系统源码

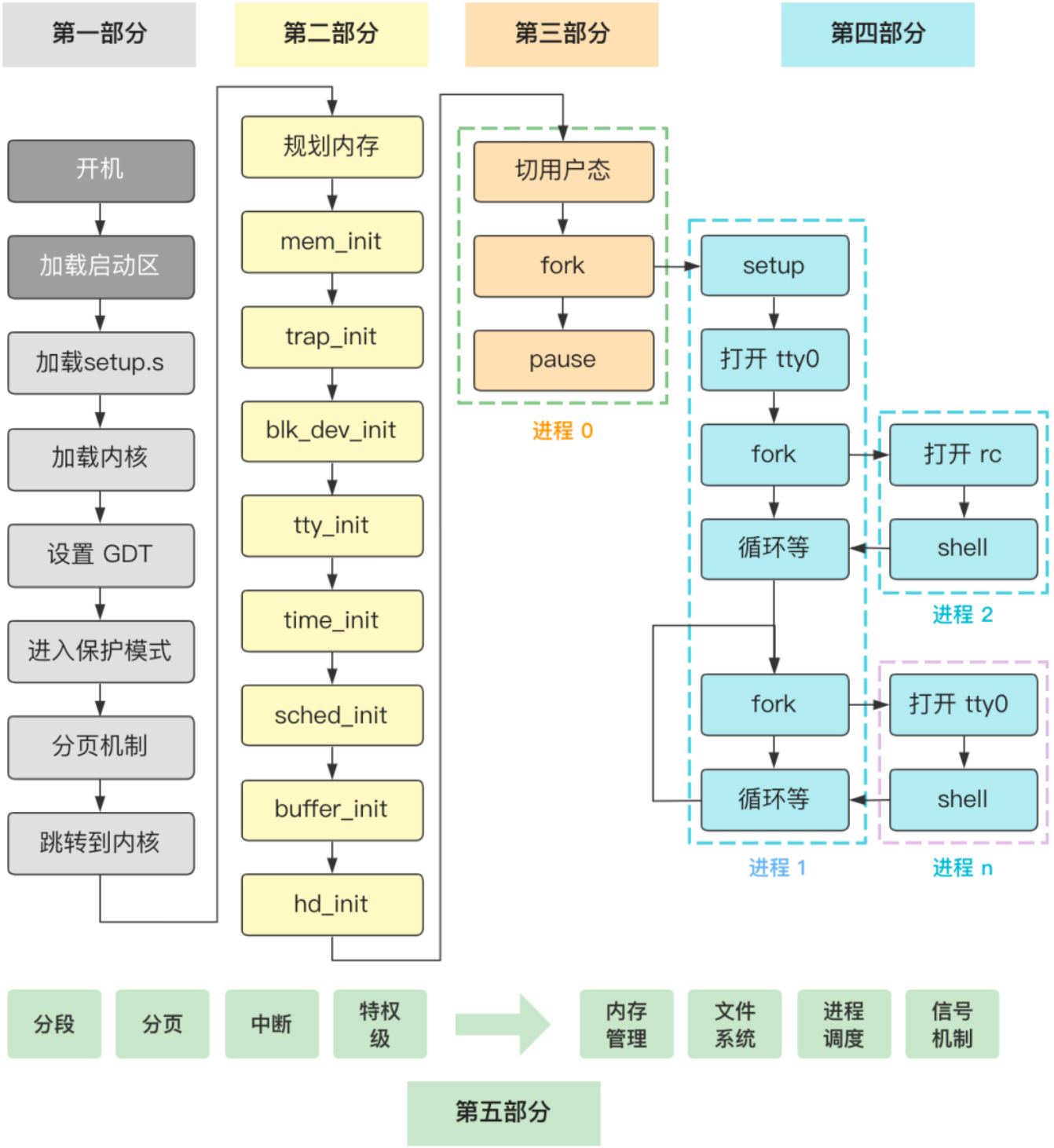
43个

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

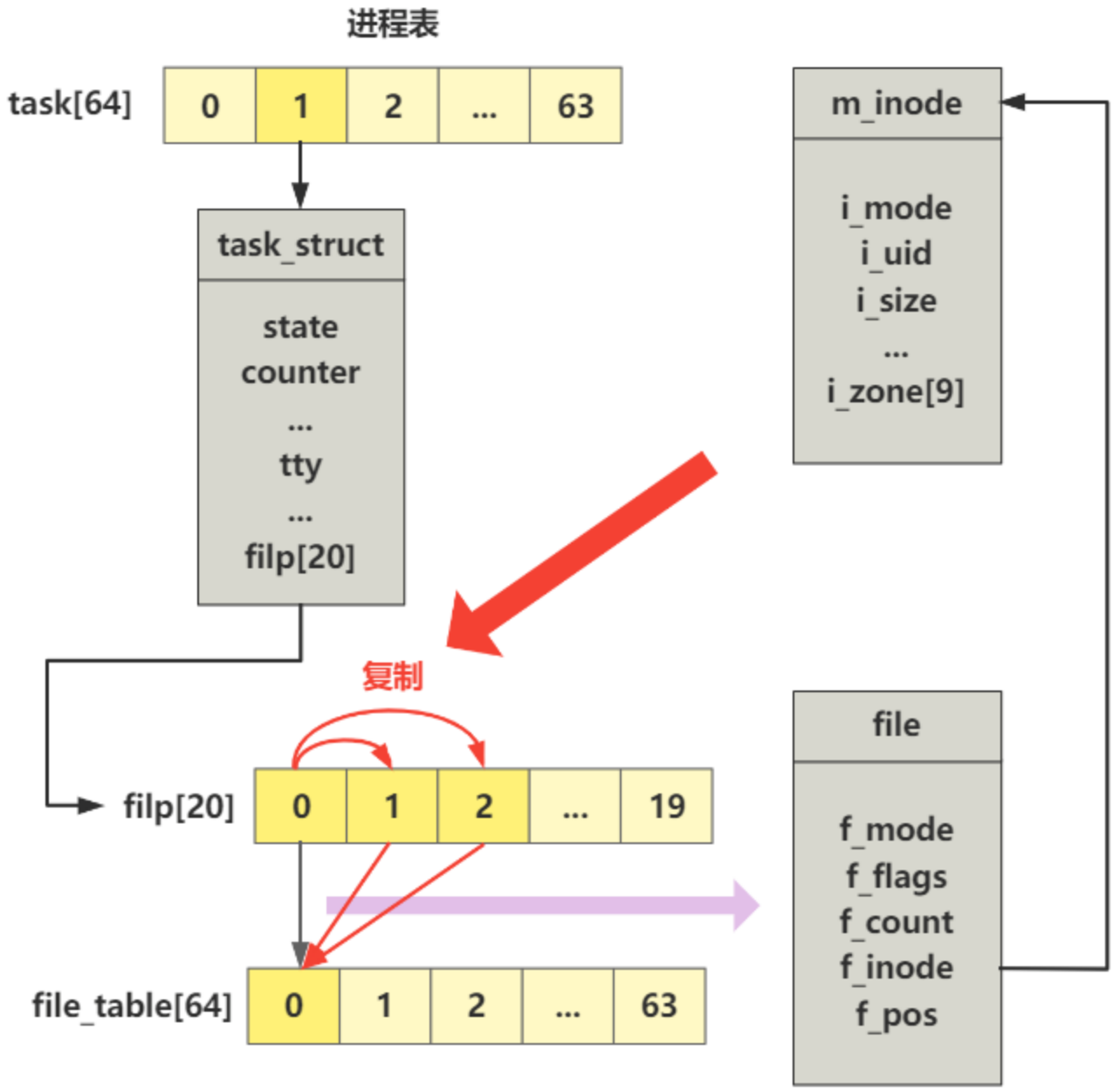
第三部分：一个新进程的诞生

第四部分：shell 程序的到来

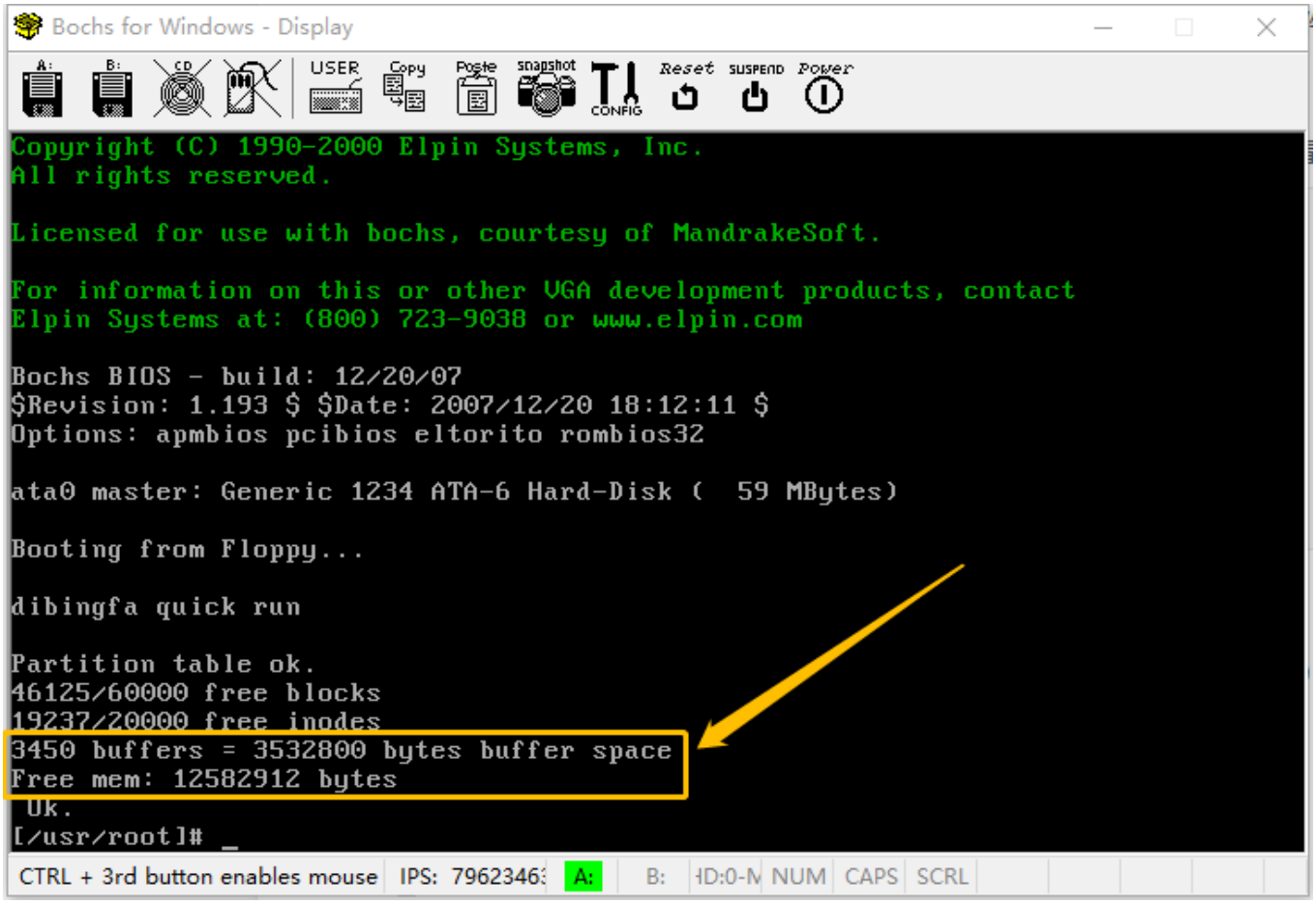
第34回 | 进程 2 的创建（本文）

------- 正文开始 -------

书接上回，上回书咱们说到，进程 1 通过 open 函数建立了与外设交互的能力，具体其实就是打开了 tty0 这个设备文件，并绑定了标准输入 0，标准输出 1 和 标准错误输出 2 这三个文件描述符。



同时我们看到源码中用 printf 函数，调用 write 函数，向 1 号文件描述符输出了字符串的效果。



到此为止，标志着进程 1 的工作基本结束了，准确说是能力建设的工作结束了，接下来就是**控制流程**和**创建新的进程**了，我们继续往下看。

```
void init(void) {
    ...
    if (!(pid=fork())) {
        close(0);
        open("/etc/rc",O_RDONLY,0);
        execve("/bin/sh",argv_rc,envp_rc);
        _exit(2);
    }
    if (pid>0)
        while (pid != wait(&i))
            /* nothing */;
    while (1) {
        if (!(pid=fork())) {
            close(0);close(1);close(2);
            setsid();
            (void) open("/dev/tty0",O_RDWR,0);
            (void) dup(0);
            (void) dup(0);
            _exit(execve("/bin/sh",argv,envp));
        }
        while (1)
            if (pid == wait(&i))
                break;
        printf("\n\rchild %d died with code %04x\n\r",pid,i);
        sync();
    }
    _exit(0);   /* NOTE! _exit, not exit() */
}
```

别急，我们一点点看，我仍然是去掉了一些错误校验的旁路分支。

```
void init(void) {
    ...
    if (!(pid=fork())) {
        close(0);
        open("/etc/rc",O_RDONLY,0);
        execve("/bin/sh",argv_rc,envp_rc);
        _exit(2);
    }
    ...
}
```

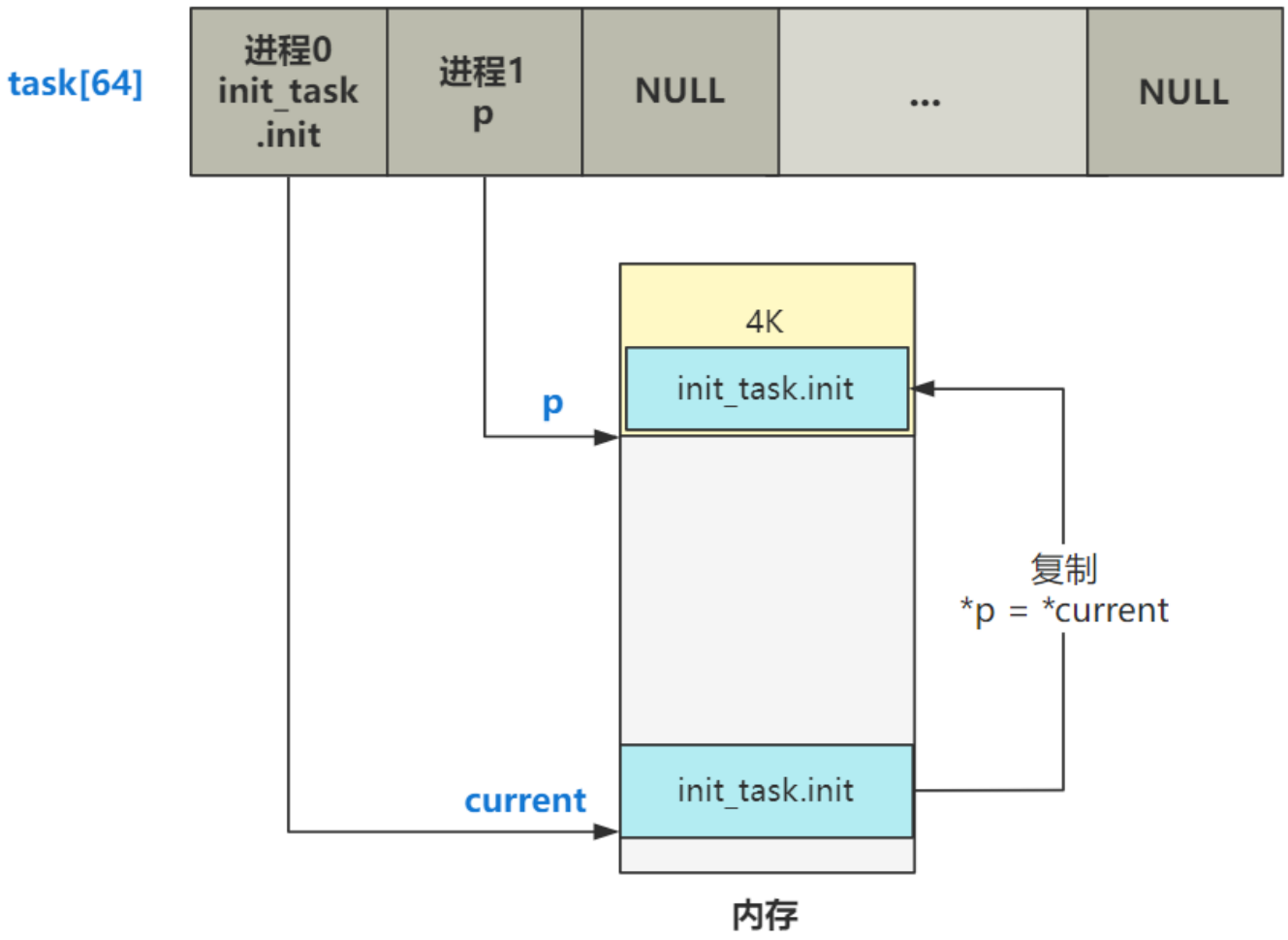
先看这个第一段，我们先尝试口述翻译一遍。

1. fork 一个新的子进程，此时就是进程 2 了。
2. 在进程 2 里**关闭（close）** 0 号文件描述符。
3. 只读形式**打开（open）** rc 文件。
4. 然后**执行（execve）** sh 程序。

听起来还蛮合逻辑的，创建进程（fork）、关闭（close）、打开（open）、执行（execve）四步走，接下来我们一点点拆解。

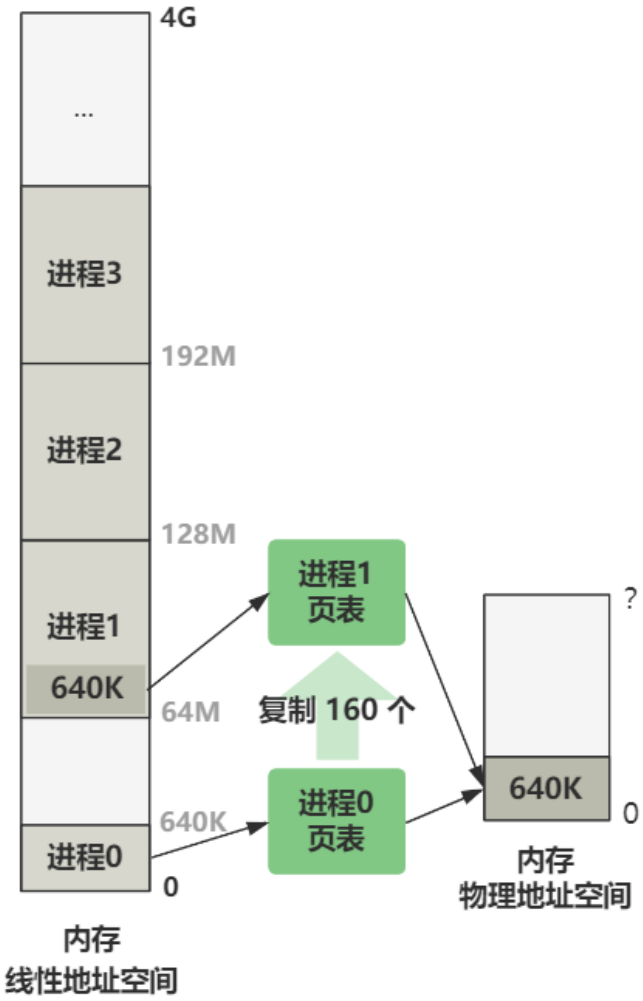
fork

fork 前面讲过了，就是将进程的 task_struct 结构进行一下复制，比如进程 0 fork 出进程 1 的时候。



之后，新进程再重写一些基本信息，包括元信息和 tss 里的寄存器信息。再之后，用 copy_page_tables 复制了一下页表（这里涉及到写时复制的伏笔）。

比如进程 0 复制出进程 1 的时候，页表是这样复制的。



而这里的进程 1 fork 出进程 2，也是同样的流程，不同之处在于两点细节：

第一点，进程 1 打开了三个文件描述符并指向了 tty0，那这个也被复制到进程 2 了，具体说来就是进程结构 task_struct 里的 flip[] 数组被复制了一份。

```
struct task_struct {
    ...
    struct file *filp[NR_OPEN];
    ...
};
```

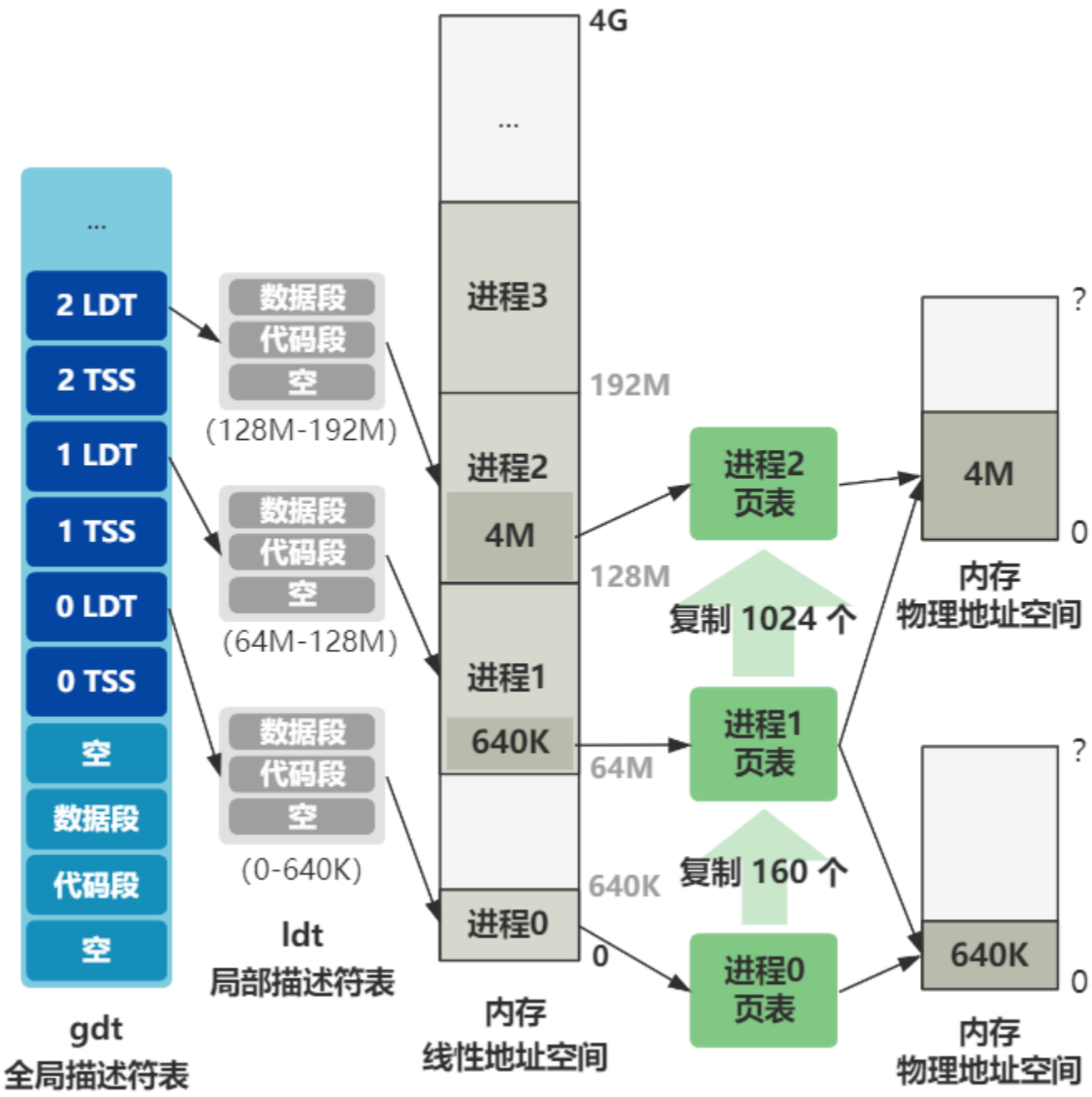
而进程 0 fork 出进程 1 时是没有复制这部分信息的，因为进程 0 没有打开任何文件。这也是刚刚说的与外设交互能力的体现，即进程 0 没有与外设交互的能力，进程 1 有，哎，其实就是这个 flip 数组里有没有东西而已嘛~

第二点，进程 0 复制进程 1 时页表的复制只有 160 项，也就是映射 640K，而之后进程的复

制，统统都是复制 1024 项，也就是映射 4M 空间。

```
int copy_page_tables(unsigned long from,unsigned long to,long size) {
    ...
    nr = (from==0)?0xA0:1024;
    ...
}
```

整体看就是如图所示。



除此之外，就没有别的区别了。

close

好了，我们继续看。

```
void init(void) {
    ...
    if (!(pid=fork())) {
        close(0);
        open("/etc/rc",O_RDONLY,0);
        execve("/bin/sh",argv_rc,envp_rc);
        _exit(2);
    }
    ...
}
```

fork 完之后，后面 if 里面的代码都是进程 2 在执行了。

close(0) 就是**关闭 0 号文件描述符**，也就是进程 1 复制过来的打开了 tty0 并作为标准输入的文件描述符，那么此时 0 号文件描述符就空出来了。

下面是 close 对应的系统调用函数，很简单。

```
int sys_close(unsigned int fd) {
    ...
    current->filp[fd] = NULL;
    ...
}
```

open

接下来 open 函数以只读形式打开了一个叫 /etc/rc 的文件，刚好占据了 0 号文件描述符的位置。

```
void init(void) {
    ...
    if (!(pid=fork())) {
        ...
        open("/etc/rc",O_RDONLY,0);
        ...
    }
    ...
}
```

这个 rc 文件表示配置文件，具体什么内容，取决于你的硬盘里这个位置处放了什么内容，与操作系统内核无关，所以我们暂且不用管。

此时，进程 2 与进程 1 几乎完全一样，只不过进程 2 通过 close 和 open 操作，将原来进程 1 的指向标准输入的 0 号文件描述符，重新指向了 /etc/rc 文件。

到目前为止，进程 2 与进程 1 的区别，仅仅是将 0 号文件描述符重新指向了 /etc/rc 文件，其他的没啥区别。

而这个 rc 文件是干嘛的，现在还不用管，肯定是后面 sh 程序要用到的，到时候在说。

execve

好，接下来进程 2 就将变得不一样了，会通过一个经典的，也是最难理解的 execve 函数调用，使自己摇身一变，成为 /bin/sh 程序继续运行，这就是下一章的重点！

```
void init(void) {
    ...
    if (!(pid=fork())) {
        ...
        execve("/bin/sh",argv_rc,envp_rc);
        ...
    }
    ...
}
```

这里就包含着操作系统究竟是如何加载并执行一个程序的原理，包括如何从文件系统中找到这个文件，如何解析一个可执行文件（在现代的 Linux 里称作 ELF 可执行文件），如何讲可执

行文件中的代码和数据加载到内存并运行。

加载到内存并运行又包含着虚拟内存等相关的知识。所以这里面的水很深，了解了这个函数，再加上 fork 函数，基本就可以把操作系统全部核心逻辑都串起来了。

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33回 | 打开终端设备文件

下一篇
第35回 | 扒开 execve 的皮

Modified on 2022-04-20

Read more