第37回 | shell 程序跑起来了

Original 闪客 低并发编程 2022-05-18 17:30 Posted on 北京

收录于合集
#操作系统源码

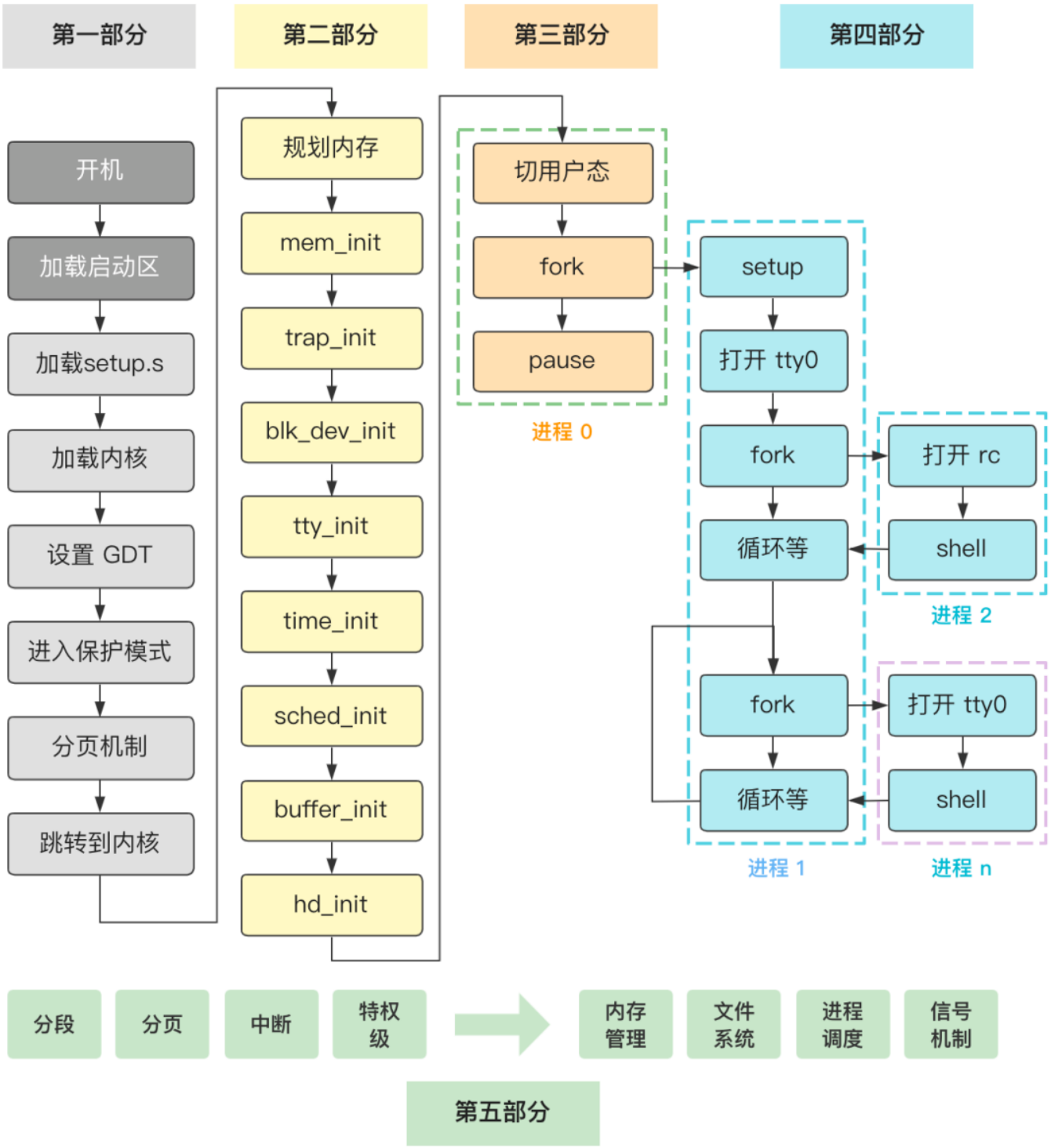
43个

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，Linux 通过缺页中断处理过程，将 /bin/sh 的代码从硬盘加载到了内存，此时便可以正式执行 shell 程序了。

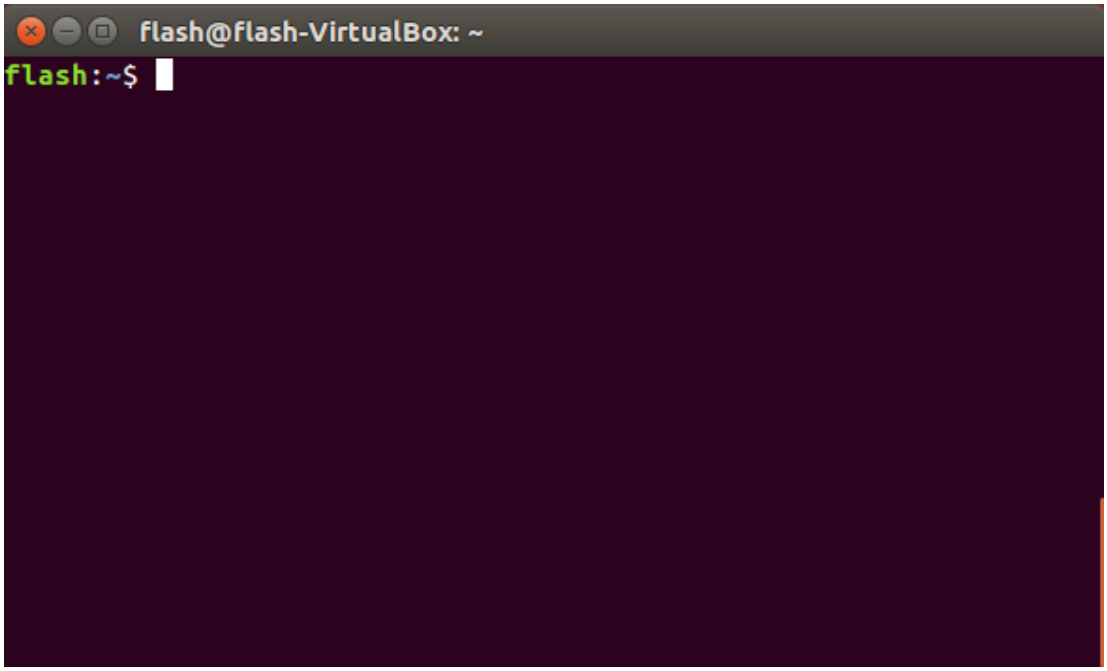
这个 **shell** 程序，也就是 Linux 0.11 中要执行的这个 **/bin/sh** 程序，它的源码并没有体现在 Linux 0.11 源码中。

也可以说，不论这个 /bin/sh 是个啥文件，哪怕只是个 hello world 程序，Linux 0.11 的启动过程中也会傻傻地去执行它。

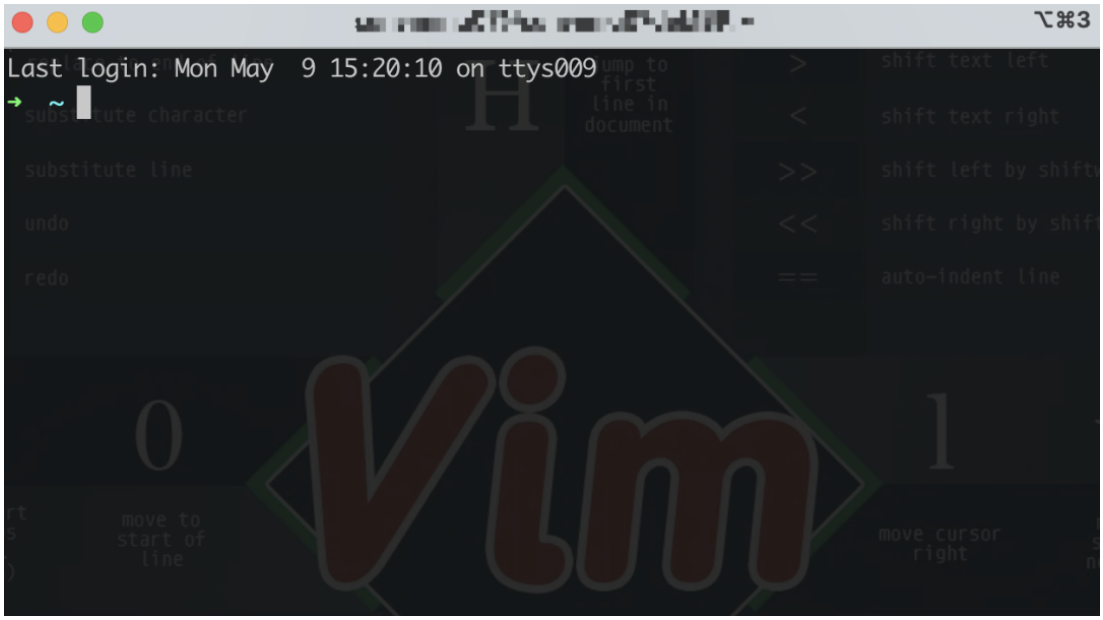
但同时，shell 又是一个我们再熟悉不过的东西了。

在我的腾讯云服务器上（用 Termius 连接），它是这个样子的。

在我的 Ubuntu 16.04 虚拟机上，它是这个样子的。



在我的 mac 电脑上，它是这个样子的。



没错，它就是我们通常说的那个命令行黑窗口。

当然 shell 只是一个标准，具体的实现可以有很多，比如在我的 Ubuntu 16.04 上，具体的 shell 实现是 bash。

```
flash:~$ echo $SHELL
/bin/bash
```

而在我的 mac 上，具体的实现是 zsh。

```
~ echo $SHELL
/bin/zsh
```

当然，默认的 shell 实现也可以手动进行设置并更改。

还有个有意思的事，shell 前面的提示符，是否可以修改呢？

我的腾讯云服务器上，提示符是

```
[root@VM-24-11-centos ~]#
```

我的 Ubuntu 虚拟机上，提示符是

```
flash:~$
```

我的 mac 电脑上更简单，提示符是

```
~
```

我现在觉得我那个腾讯云服务器上的提示符太长了怎么办？我们先查看一个变量 PS1 的值

```
[root@VM-24-11-centos ~]# echo $PS1
[\u@\h \W]\$
```

然后，我们直接把这个值给改了。

```
[root@VM-24-11-centos ~]# echo $PS1
[\u@\h \W]\$
[root@VM-24-11-centos ~]# PS1=[呵呵呵]
[呵呵呵]
```

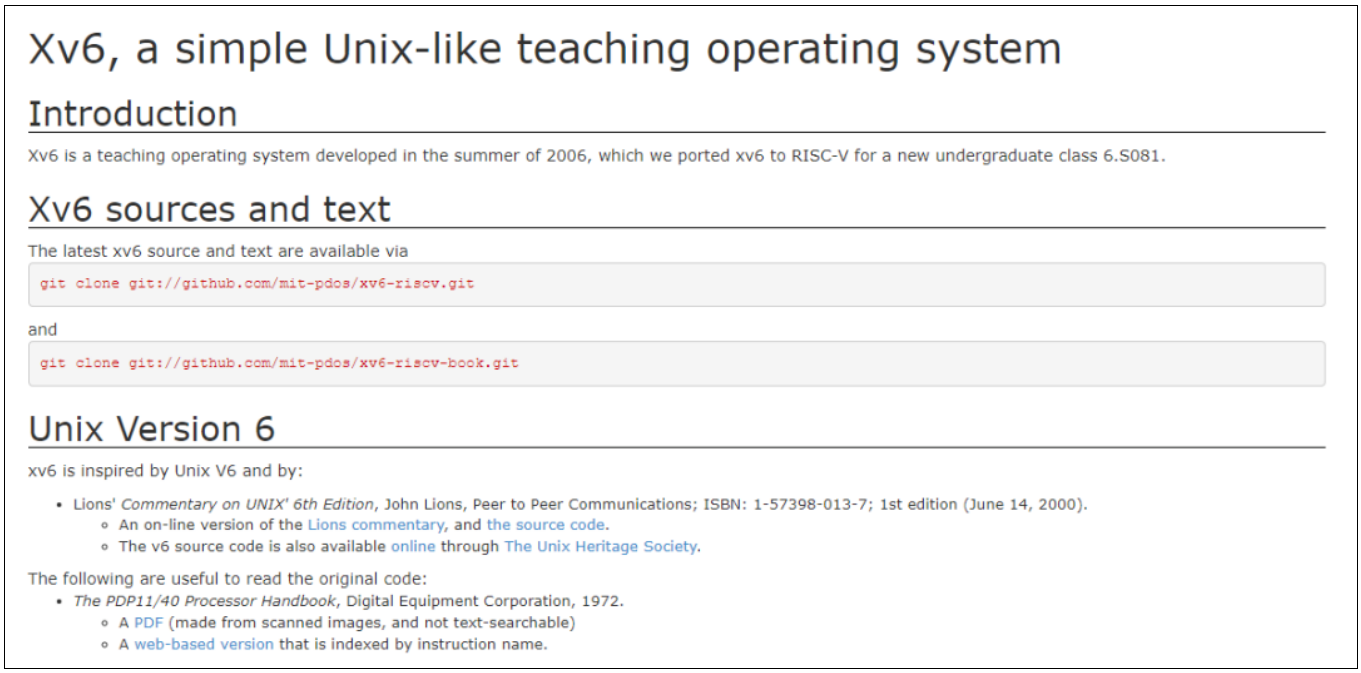
可以看到神奇的事情发生了，前面的提示符变成了我们自己定义的样子。

其实我就想说，shell 程序也仅仅是个程序而已，它的输出，它的输入，它的执行逻辑，是完全可以通过阅读程序源码来知道的，和一个普通的程序并没有任何区别。

好了，接下来我们就阅读一下 shell 程序的源码，只需要找到它的一个具体实现即可。但是 bash，zsh 等实现都过于复杂，很多东西对于我们学习完全没必要。

所以这里我通过一个非常非常精简的 shell 实现，即 **xv6** 里的 shell 实现为例，来进行讲解。

xv6 是一个非常非常经典且简单的操作系统，是由麻省理工学院为操作系统工程的课程开发的一个**教学目的的操作系统**，所以非常适合操作系统的学习。



Xv6, a simple Unix-like teaching operating system

Introduction

Xv6 is a teaching operating system developed in the summer of 2006, which we ported xv6 to RISC-V for a new undergraduate class 6.S081.

Xv6 sources and text

The latest xv6 source and text are available via

```
git clone git://github.com/mit-pdos/xv6-riscv.git
```

and

```
git clone git://github.com/mit-pdos/xv6-riscv-book.git
```

Unix Version 6

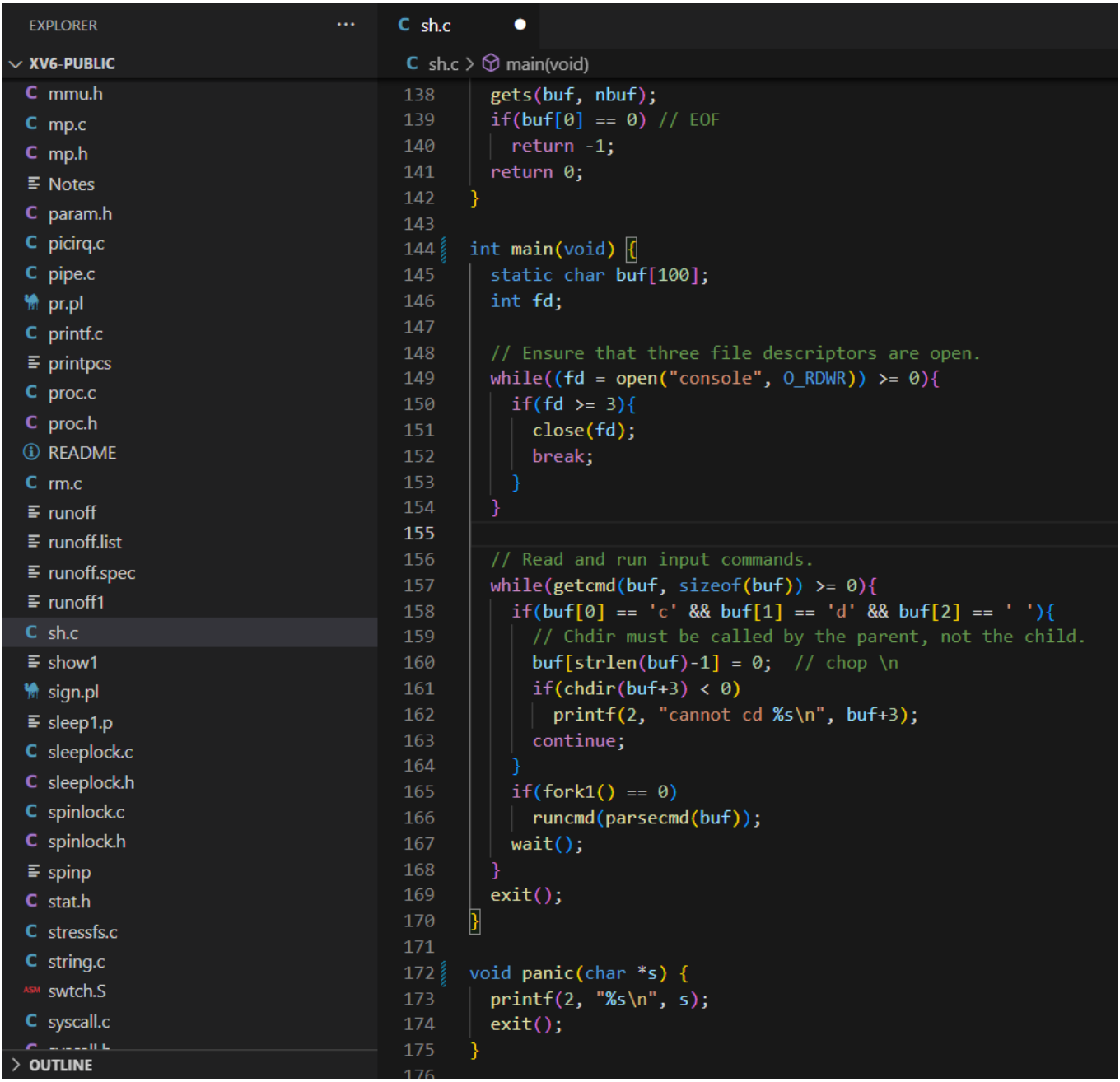
xv6 is inspired by Unix V6 and by:

- Lions' *Commentary on UNIX' 6th Edition*, John Lions, Peer to Peer Communications; ISBN: 1-57398-013-7; 1st edition (June 14, 2000).
  - An on-line version of the Lions commentary, and the source code.
  - The v6 source code is also available online through The Unix Heritage Society.

The following are useful to read the original code:

- *The PDP11/40 Processor Handbook*, Digital Equipment Corporation, 1972.
  - A PDF (made from scanned images, and not text-searchable)
  - A web-based version that is indexed by instruction name.

而在它的源代码中，又恰好实现了一个简单的 shell 程序，所以阅读它的代码，对我们这个系列课程来说，简直再合适不过了。



看到没，甚至在这么一个小小的截图里，已经可以完整展示 sh.c 里全部的 main 方法代码了。

但我仍然十分贪婪，即便是这么短的代码，我也帮你把一些多余的校验逻辑去掉，再去掉关于 cd 命令的特殊处理分支，来一个最干净的版本。

```
// xv6-public sh.c
int main(void) {
    static char buf[100];
    // 读取命令
    while(getcmd(buf, sizeof(buf)) >= 0){
        // 创建新进程
        if(fork() == 0)
            // 执行命令
            runcmd(parsecmd(buf));
        // 等待进程退出
        wait();
    }
}
```

看，shell 程序变得异常简单了！

总得来说，shell 程序就是个死循环，它永远不会自己退出，除非我们手动终止了这个 shell 进程。

在死循环里面，shell 就是不断读取（**getcmd**）我们用户输入的命令，创建一个新的进程（**fork**），在新进程里执行（**runcmd**）刚刚读取到的命令，最后等待（**wait**）进程退出，再次进入读取下一条命令的循环中。

由此你是不是也感受到了 xv6 源码的简单之美，真的是见名知意，当你跟我走完这个 Linux 0.11 之旅后，再去阅读 xv6 的源码你会觉得非常舒服，因为 Linux 0.11 很多地方都用了非常骚的编码技巧，使得理解起来很困难，谁让 Linus 这么特立独行呢。

我们之前说过 shell 就是不断 **fork + execve** 完成执行一个新程序的功能的，那 execve 在哪呢？

那我们就要看执行命令的 runcmd 代码了。

```
void runcmd(struct cmd *cmd) {
    ...
    struct execcmd ecmd = (struct execcmd*)cmd;
    ...
    exec(ecmd->argv[0], ecmd->argv);
    ...
}
```

这里我又省略了很多代码，比如遇到管道命令 PIPE，遇到命令集合 LIST 时的处理逻辑，我们仅仅看单纯执行一条命令的逻辑。

可以看到，就是简简单单调用了个 exec 函数，这个 exec 是 xv6 代码里的名字，在 Linux 0.11 里就是我们在 第35回 | execve 加载并执行 shell 程序 里讲的 execve 函数。

shell 执行一个我们所指定的程序，就和我们在 Linux 0.11 里通过 fork + execve 函数执行了 /bin/sh 程序是一个道理。

你看，fork 和 execve 函数你一旦懂了，shell 程序的原理你就直接秒懂了。而 fork 和 execve 函数的原理，其实如果你非常熟练地掌握中断、虚拟内存、文件系统、进程调度等更为底层的基础知识，其实也不难理解。

所以，根基真的很重要，本回已经到操作系统启动流程的最后一哆嗦了，如果你现在感觉十分混乱，最好的办法就是，不断去啃之前那些你认为"无聊的"、"没用的"章节。

好了，今天的 shell 就到这里了，毕竟我们是讲 Linux 0.11 核心流程的系列，不必过多深入 shell 这个应用程序。

接下来有个问题，shell 程序执行了，操作系统就结束了么？

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。

收录于合集 #操作系统源码 43

上一篇
第36回 | 缺页中断

下一篇
第38回 | 操作系统启动完毕！

Read more

People who liked this content also liked

从Go log库到Zap，怎么打造出好用又实用的Logger

Golang技术分享

记住这两兄弟，他们可能是 Web 史上最大的错误

进击的Coder

外部函数如何访问其它类的私有成员

程序喵大人