

## 60 | 搭建操作系统实验环境（上）：授人以鱼不如授人以渔

2019-08-14 刘超

趣谈Linux操作系统

[进入课程 >](#)



讲述：刘超

时长 14:48 大小 13.57M



操作系统的理论部分我们就讲完了，但是计算机这门学科是实验性的。为了更加深入地了解操作系统的本质，我们必须能够做一些上手实验。操作系统的实验，相比其他计算机课程的实验要更加复杂一些。

我们做任何实验，都需要一个实验环境。这个实验环境要搭建在操作系统之上，但是，我们这个课程本身就是操作系统实验，难不成要自己 debug 自己？到底该咋整呢？

我们有一个利器，那就是 qemu 啊，不知道你还记得吗？它可以在操作系统之上模拟一个操作系统，就像一个普通的进程。那我们是否可以像 debug 普通进程那样，通过 qemu 来 debug 虚拟机里面的操作系统呢？

这一节和下一节，我们就按照这个思路，来试试看，搭建一个操作系统的实验环境。


运行一个 qemu 虚拟机，首先我们要有一个虚拟机的镜像。咱们在[虚拟机](#)那一节，已经制作了一个虚拟机的镜像。假设我们要基于 [ubuntu-18.04.2-live-server-amd64.iso](#)，它对应的内核版本是 linux-source-4.15.0。

当时我们启动虚拟机的过程很复杂，设置参数的时候也很复杂，以至于解析这些参数就花了我们一章的时间。所以，这里我介绍一个简单的创建和管理虚拟机的方法。

在[CPU 虚拟化](#)那一节，我留过一个思考题，OpenStack 是如何创建和管理虚拟机的？当时我给了你一个提示，就是用 libvirt。没错，这一节，我们就用 libvirt 来创建和管理虚拟机。


## 创建虚拟机

首先，在宿主机上，我们需要一个网桥。我们用下面的命令创建一个网桥，并且设置一个 IP 地址。

 复制代码


```
1 brctl addbr br0
2 ip link set br0 up
3 ifconfig br0 192.168.57.1/24
```

为了访问外网，这里还需要设置 /etc/sysctl.conf 文件中 net.ipv4.ip\_forward=1 参数，并且执行以下的命令，设置 NAT。

 复制代码


```
1 iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

接下来，就要创建虚拟机了。这次我们就不再一个个指定虚拟机启动的参数，而是用 libvirt。首先，使用下面的命令，安装 libvirt。

 复制代码

```
1 apt-get install libvirt-bin
2 apt-get install virtinst
```

libvirt 管理 qemu 虚拟机，是基于 XML 文件，这样容易维护。


 复制代码

```
1 <domain type='qemu'>
2   <name>ubuntutest</name>
3   <uuid>0f0806ab-531d-6134-5def-c5b4955292aa</uuid>
4   <memory unit='GiB'>4</memory>
5   <currentMemory unit='GiB'>4</currentMemory>
6   <vcpu placement='static'>2</vcpu>
7   <os>
8     <type arch='x86_64' machine='pc-i440fx-trusty'>hvm</type>
9     <boot dev='hd' />
10  </os>
11  <features>
12    <acpi />
13    <apic />
14    <pae />
15  </features>
16  <clock offset='utc' />
17  <on_poweroff>destroy</on_poweroff>
18  <on_reboot>restart</on_reboot>
19  <on_crash>restart</on_crash>
20  <devices>
21    <emulator>/usr/bin/qemu-system-x86_64</emulator>
22    <disk type='file' device='disk'>
23      <driver name='qemu' type='qcow2' />
24      <source file='/mnt/vdc/ubuntutest.img' />
25      <target dev='vda' bus='virtio' />
26    </disk>
27    <controller type='pci' index='0' model='pci-root' />
28    <interface type='bridge'>
29      <mac address='fa:16:3e:6e:89:ce' />
30      <source bridge='br0' />
31      <target dev='tap1' />
32      <model type='virtio' />
33    </interface>
34    <serial type='pty'>
35      <target port='0' />
36    </serial>
37    <console type='pty'>
38      <target type='serial' port='0' />
39    </console>
40    <graphics type='vnc' port='-1' autoport='yes' listen='0.0.0.0'>
41      <listen type='address' address='0.0.0.0' />
42    </graphics>
43    <video>
44      <model type='cirrus' />
45    </video>
```

```
46     </devices>
47 </domain>
```


在这个 XML 文件中，/mnt/vdc/ubuntu-test.img 就是虚拟机的镜像，br0 就是我们创建的网桥，连接到网桥上的网卡 libvirt 会自动帮我们创建。

接下来，需要将这个 XML 保存为 domain.xml，然后调用下面的命令，交给 libvirt 进行管理。

 复制代码


```
1 virsh define domain.xml
```

接下来，运行 virsh list --all，我们就可以看到这个定义好的虚拟机了，然后我们调用 virsh start ubuntu-test，启动这个虚拟机。

 复制代码

```
1 # virsh list
2  Id      Name                                State
3  -----
4  1       ubuntu-test                      running
```

我们可以通过 ps 查看 libvirt 启动的 qemu 进程。这个命令行是不是很眼熟？我们之前花了一章来讲解。如果不记得了，你可以回去看看前面的内容。


 复制代码

```
1 # ps aux | grep qemu
2 libvirt+ 9343 85.1 34.7 10367352 5699400 ?    Sl   Jul27 1239:18 /usr/bin/qemu-system-
```

从这里，我们可以看到，VNC 的设置为 0.0.0.0:0。我们可以用 VNCViewer 工具登录到这个虚拟机的界面，但是这样实在是太麻烦了，其实 virsh 有一个特别好的工具，但是需要在虚拟机里面配置一些东西。


在虚拟机里面，我们修改 /boot/grub/ 里面的两个文件，一个是 grub.cfg，另一个是 menu.lst，这里面就是咱们在[系统初始化](#)的时候，讲过的那个启动列表。

在 grub.cfg 中，在 submenu 'Advanced options for Ubuntu' 这一项，在这一行的 linux /boot/vmlinuz-4.15.0-55-generic root=UUID=470f3a42-7a97-4b9d-aaa0-26deb3d234f9 ro console=ttyS0 maybe-ubiquity 中，加上了 console=ttyS0。

 复制代码


```
1 submenu 'Advanced options for Ubuntu' $menuentry_id_option 'gnulinux-advanced-470f3a42-7a97-4b9d-aaa0-26deb3d234f9'
2     menuentry 'Ubuntu, with Linux 4.15.0-55-generic' --class ubuntu --class gnu-linux --class os {
3         recordfail
4         load_video
5         gfxmode $linux_gfx_mode
6         insmod gzio
7         if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
8         insmod part_gpt
9         insmod ext2
10        set root='hd0,gpt2'
11        if [ x$feature_platform_search_hint = xy ]; then
12            search --no-floppy --fs-uuid --set=root --hint-bios=hd0,gpt2 --hint-efi=hd0,gpt2 --hint-usb=usb-lun:0:0:1 --no-uuid
13        else
14            search --no-floppy --fs-uuid --set=root 470f3a42-7a97-4b9d-aaa0-26deb3d234f9
15        fi
16        echo 'Loading Linux 4.15.0-55-generic ...'
17        linux /boot/vmlinuz-4.15.0-55-generic root=UUID=470f3a42-7a97-4b9d-aaa0-26deb3d234f9 ro console=ttyS0 maybe-ubiquity
18        echo 'Loading initial ramdisk ...'
19        initrd /boot/initrd.img-4.15.0-55-generic
20    }
```

在 menu.lst 文件中，在 Ubuntu 18.04.2 LTS, kernel 4.15.0-55-generic 这一项，在 kernel /boot/vmlinuz-4.15.0-55-generic root=/dev/hda1 ro console=hvc0 console=ttyS0 这一行加入 console=ttyS0。

 复制代码


```
1 title          Ubuntu 18.04.2 LTS, kernel 4.15.0-55-generic
2 root           (hd0)
3 kernel         /boot/vmlinuz-4.15.0-55-generic root=/dev/hda1 ro console=hvc0 console=ttyS0
4 initrd         /boot/initrd.img-4.15.0-55-generic
```

接下来，我们重启虚拟机，重启后上面的配置就起作用了。这时候，我们可以通过下面的命令，进入机器的控制台，可以不依赖于 SSH 和 IP 地址进行登录。

 复制代码

```
1 # virsh console ubuntutest
2 Connected to domain ubuntutest
3 Escape character is ^]
```

下面，我们可以配置这台机器的 IP 地址了。对于 ubuntu-18.04 来讲，IP 地址的配置方式为修改 `/etc/netplan/50-cloud-init.yaml` 文件。

 复制代码

```
1 network:
2   ethernets:
3     ens3:
4       addresses: [192.168.57.100/24]
5       gateway4: 192.168.57.1
6       dhcp4: no
7       nameservers:
8         addresses: [8.8.8.8,114.114.114.114]
9       optional: true
10  version: 2
```

然后，我们可以通过 `netplan apply`，让配置生效，这样，虚拟机里面的 IP 地址就配置好了。现在，我们应该能 ping 得通公网的一个网站了。

虚拟机就此创建好了，接下来我们需要下载源代码重新编译。


## 下载源代码

首先，我们先下载源代码。

 复制代码

```
1 apt-get install linux-source-4.15.0
```

这行命令会将代码下载到 /usr/src/ 目录下，我们可以通过下面的命令解压缩。

 复制代码


```
1 tar vxkf linux-source-4.15.0.tar.bz2
```

至此，路径 /usr/src/linux-source-4.15.0 下，就是解压好的内核代码。

准备工作都做好了。这一节，我们先来做第一个实验，也就是，在原有内核代码的基础上加一个我们自己的系统调用。

在哪里加代码呢？如果你忘了，请出门左转，回顾一下[系统调用](#)那一节。


第一个要加的地方是 arch/x86/entry/syscalls/syscall\_64.tbl。这里面登记了所有的系统调用号以及相应的处理函数。

 复制代码

```
1 332      common  statx              sys_statx
2 333      64      sayhelloworld    sys_sayhelloworld
```

在这里，我们找到 332 号系统调用 sys\_statx，然后照猫画虎，添加一个 sys\_sayhelloworld，这里我们只添加 64 位操作系统的。


第二个要加的地方是 include/linux/syscalls.h，也就是系统调用的头文件，然后添加一个系统调用的声明。

 复制代码

```
1 asmlinkage long sys_statx(int dfd, const char __user *path, unsigned flags,
2                          unsigned mask, struct statx __user *buffer);
3
4 asmlinkage int sys_sayhelloworld(char * words, int count);
```

同样，我们找到 `sys_statx` 的声明，照猫画虎，声明一个 `sys_sayhelloworld`。其中，`words` 参数是用户态传递给内核态的文本的指针，`count` 是数目。

第三个就是对于这个系统调用的实现，方便起见，我们不再用 `SYSCALL_DEFINE` 系列的宏来定义了，直接在 `kernel/sys.c` 中实现。


 复制代码

```
1 asmlinkage int sys_sayhelloworld(char * words, int count){
2     int ret;
3     char buffer[512];
4     if(count >= 512){
5         return -1;
6     }
7     copy_from_user(buffer, words, count);
8     ret=printk("User Mode says %s to the Kernel Mode!", buffer);
9     return ret;
10 }
```

接下来就要开始编译内核了。

## 编译内核

编译之前，我们需要安装一些编译要依赖的包。

 复制代码

```
1 apt-get install libncurses5-dev libssl-dev bison flex libelf-dev gcc make openssl libc6
```


首先，我们要定义编译选项。

 复制代码

```
1 make menuconfig
```


然后，我们能通过选中下面的选项，激活 `CONFIG_DEBUG_INFO` 和 `CONFIG_FRAME_POINTER` 选项。



 复制代码


```
1 Kernel hacking --->
2 Compile-time checks and compiler options --->
3 [*] Compile the kernel with debug info
4 [*] Compile the kernel with frame pointers
```

选择完毕之后，配置会保存在.config 文件中。如果我们打开看，能看到这样的配置：

 复制代码

```
1 CONFIG_FRAME_POINTER=y
2 CONFIG_DEBUG_INFO=y
```


接下来，我们编译内核。

 复制代码

```
1 nohup make -j8 > make1.log 2>&1 &
2 nohup make modules_install > make2.log 2>&1 &
3 nohup make install > make3.log 2>&1 &
```

这是一个非常长的过程，请耐心等待，可能需要数个小时，因而这里用了 nohup，你可以去干别的事情。

当编译完毕之后，grub 和 menu.lst 都会发生改变。例如，grub.conf 里面会多一个新内核的项。

 复制代码

```
1 submenu 'Advanced options for Ubuntu' $menuentry_id_option 'gnulinux-advanced-470f3a42-1
2     menuentry 'Ubuntu, with Linux 4.15.18' --class ubuntu --class gnu-linux --class
3         recordfail
4         load_video
5         gfxmode $linux_gfx_mode
6         insmod gzio
7         if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
8         insmod part_gpt
9         insmod ext2
10        if [ x$feature_platform_search_hint = xy ]; then
```

```

11         search --no-floppy --fs-uuid --set=root 470f3a42-7a97-4b9d-aaa0-26del
12     else
13         search --no-floppy --fs-uuid --set=root 470f3a42-7a97-4b9d-aaa0-26deb:
14     fi
15     echo 'Loading Linux 4.15.18 ...'
16     linux /boot/vmlinuz-4.15.18 root=UUID=470f3a42-7a97-4b9d-aaa0-26deb3d:
17     echo 'Loading initial ramdisk ...'
18     initrd /boot/initrd.img-4.15.18
19 }

```

例如，menu.lst 也多了新的内核的项。

 复制代码


```

1 title          Ubuntu 18.04.2 LTS, kernel 4.15.18
2 root           (hd0)
3 kernel         /boot/vmlinuz-4.15.18 root=/dev/hda1 ro console=hvc0 console=ttyS0
4 initrd         /boot/initrd.img-4.15.18

```

别忘了，这里面都要加上 console=ttyS0。

下面，我们要做的就是重启虚拟机。进入的时候，会出现 GRUB 界面。我们选择 Ubuntu 高级选项，然后选择第一项进去，通过 uname 命令，我们就进入了新的内核。


 复制代码

```

1 # uname -a
2 Linux popsuper 4.15.18 #1 SMP Sat Jul 27 13:43:42 UTC 2019 x86_64 x86_64 x86_64 GNU/Lin

```

进入新的系统后，我们写一个测试程序。

 复制代码


```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <linux/kernel.h>
5 #include <sys/syscall.h>
6 #include <string.h>
7

```

```
8 int main ()
9 {
10     char * words = "I am liuchao from user mode.";
11     int ret;
12     ret = syscall(333, words, strlen(words)+1);
13     printf("return %d from kernel mode.\n", ret);
14     return 0;
15 }
```

然后，我们能利用 gcc 编译器编译后运行。如果我们查看日志 /var/log/syslog，就能够看到里面打印出来下面的日志，这说明我们的系统调用已经添加成功了。

 复制代码

```
1 Aug  1 06:33:12 popsuper kernel: [ 2048.873393] User Mode says I am liuchao from user m
```

## 总结时刻

这一节是一节实战课，我们创建了一台虚拟机，在里面下载源代码，尝试修改了 Linux 内核，添加了一个自己的系统调用，并且进行了编译并安装了新内核。如果你按照这个过程做下来，你会惊喜地发现，原来令我们敬畏的内核，也是能够加以干预，为我而用的呢。没错，这就是你开始逐渐掌握内核的重要一步。

## 课堂练习

这一节的课堂练习，希望你能够按照整个过程，一步一步操作下来。毕竟看懂不算懂，做出来才算入门啊。

欢迎留言和我分享你的疑惑和见解，也欢迎你收藏本节内容，反复研读。你也可以把今天的内容分享给你的朋友，和他一起学习、进步。

# 趣谈 Linux 操作系统

像故事一样的操作系统入门课

刘超

网易杭州研究院

云计算技术部首席架构师



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 59 | 数据中心操作系统：上市敲钟

下一篇 61 | 搭建操作系统实验环境（下）：授人以鱼不如授人以渔

## 精选留言 (6)

写留言



LDxy

2019-08-15

可以在虚拟机里运行虚拟机吗？

展开 ▾



许童童

2019-08-14

跟着老师一起动手，实战。

展开 ▾



郭江伟

2019-08-14

请问下老师 每次修改内核都需要全部编译吗？可以部分编译吗？



**leslie**

2019-08-14

发现老师的课如老师自己介绍的学习方法一样：不是一遍就能学懂的，跟着做跟着反思；要第二遍或者第三遍才能理解和明白老师所讲所授的知识。

看来所谓的第一遍或者第二遍第三遍只是大概：其实应当是三个阶段/层次；努力坚持努力学习，希望多遍之后能尽力掌握其6-8成。

展开 ▾



**大王叫我来巡山**

2019-08-14

当年上课的时候只是给了个文档，让增加系统调用，其实并不明白，终于看明白了



**Marshall**

2019-08-14

后期准备跟着老师动手一下

展开 ▾

