

目录

课程目标	2
实验环境	2
实验步骤	2
功能目标实现	22

课程目标

完成最新版本 Linux kernel 内核及其配套的 RAMDisk 文件系统定制工作。

大小要求：内核文件<3M,initrd.img<24M

功能要求：

1. 通过 U 盘或 PXE 网络启动加载 kernel 和 img 启动进行验证
2. 支持多用户登录（console 界面和 ssh 网络方式）
3. 系统支持通过 ssh 方式访问其他机器
4. 可挂载 U 盘
5. 可访问机器上的 windows 分区（ntfs-3g fs 支持）

实验环境

电脑：MacBook Pro M2（aarch64 指令集）

操作系统：macOS 14.0

虚拟机软件：VMware Fision Player 13.0.2

Linux 标准系统：Ubuntu 22.04，内核版本 5.15.0

供裁剪用的最新内核版本：6.3.6

完整版代码：https://github.com/ShenMuyuan/custom_linux

实验步骤

1. 准备工作：Grub 新增启动项

参考/boot/grub/grub.cfg，在/etc/grub.d/40_custom 中增加所需的启动项（我这里增加了三项，重启、关机和进入 RAM 小系统，如图 1）：

```
menuentry 'Restart My Computer' {  
    reboot  
}  
  
menuentry 'Shutdown My Computer' {  
    halt  
}  
  
menuentry 'Minimal initramfs and kernel by SMY' {  
    linux    /smy_vmlinuz.gz  
    initrd  /smy_initrd.gz  
}
```

图1 增加的 menuentry

注：关于启动用到的 Linux 内核，我最早使用的就是自己编译的而不是标准系统提供的，因为发现直接使用标准系统的内核会无法启动。

之后的任何版本，在启动时可以看到如图 2 所示的 grub 选项：

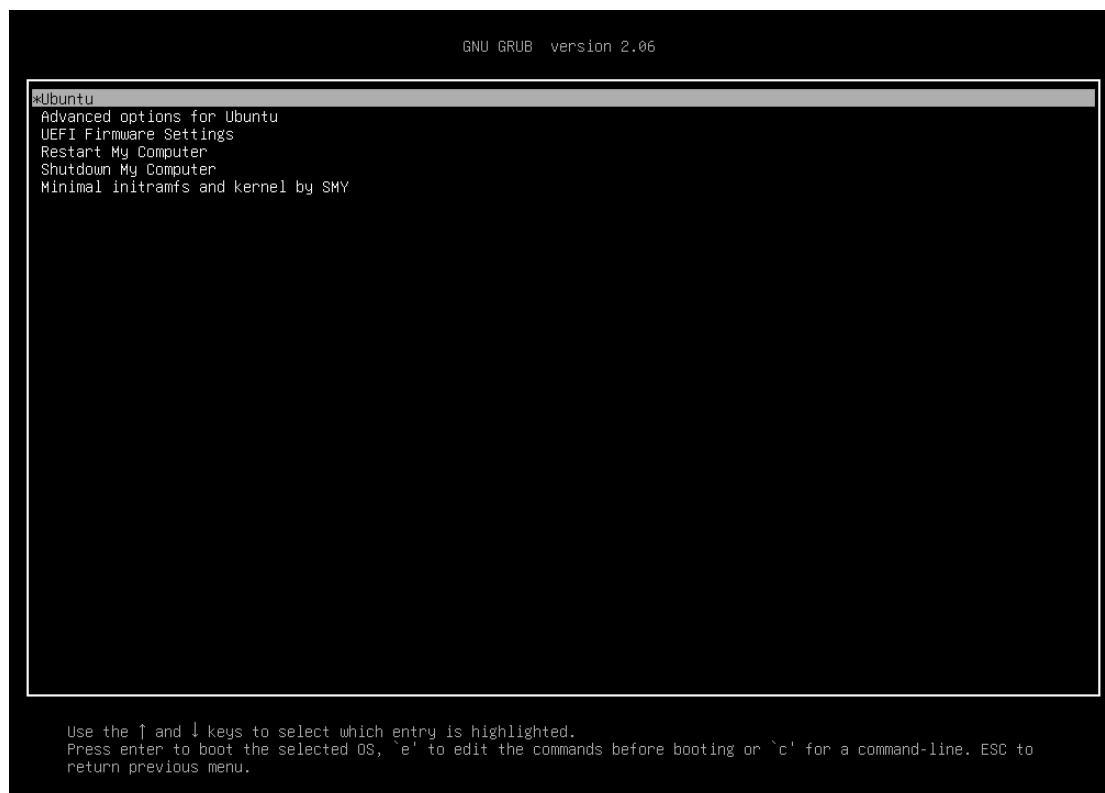


图2 grub 选项

2. v0.5: 在小系统中获得一个 shell

小系统执行的是其根目录的 init 脚本，获得 shell 也就是让 init 调用 bash 程序。要使得 bash 程序正确运行，需要拷贝这个可执行文件本身以及依赖的动态库。为了方便，编写了 add_file、add_commands 两个 shell 函数，分别用于添加文件和添加命令，如图 3、4 所示。

```
1  #!/bin/bash
2
3  # Usage: add_file /full/path/to/file
4  # add single file only
5
6  add_file() {
7      if [[ ! -e $1 ]]; then
8          echo "Error: wrong file $1"
9          exit
10     fi
11
12     root=${1%/*}    # removes the last forward slash and everything after it
13
14     # add the file to the same directory structure in current directory
15     if [[ ! -e "$(pwd)$1" ]]; then
16         if [[ ! -d "$(pwd)$root" ]]; then
17             mkdir -p "$(pwd)$root"
18         fi
19         cp "$1" "$(pwd)$1"
20     fi
21 }
```

图3 add_file 函数

```

1  #!/bin/bash
2
3  # Usage: add_commands cmd1 cmd2 ...
4  # can be name or path
5
6  . utils/add_file.sh
7
8  add_commands() {
9      for cmd in "$@"; do
10         if [[ $cmd =~ ^\| ]]; then # cmd is full path
11             if [[ -e "$cmd" ]]; then
12                 path=$cmd
13             else
14                 echo "Error: wrong command $cmd"
15                 exit
16             fi
17         else # cmd is a name
18             if ! which "$cmd"; then
19                 echo "Error: wrong command $cmd"
20                 exit
21             fi
22             path=$(which "$cmd")
23         fi
24         echo "Adding command: $path"
25         add_file "$path"
26
27         for line in $(ldd "$path"); do # split words by space and newline
28             if [[ $line =~ ^\| ]]; then
29                 echo "Adding dependency library: $line"
30                 add_file "$line"
31             fi
32         done
33     done
34 }

```

图4 add_commands 函数

小系统中，发现键盘无法输入。在标准系统查看 lsmod，与键盘相关的有 xhci_pci 和 xhci_pci_renesas 两项（后者被前者依赖，如图 5）

```

root@jammy:/home/smy/dev/kernel# lsmod | grep xhci
xhci_pci                24576  0
xhci_pci_renesas        24576  1 xhci_pci

```

图5 键盘输入相关模块

而在 Linux kernel 的 menuconfig 中搜索可知，二者默认也是编译为模块（如图 6）：

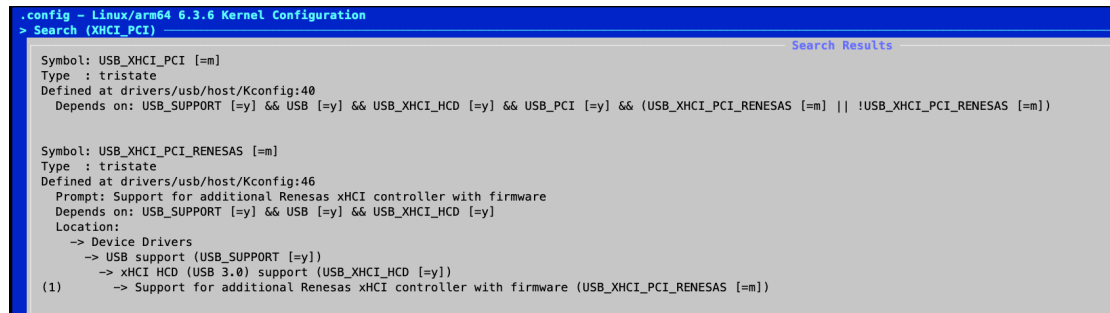


图6 kernel config 中键盘输入相关模块

从编译好的 kernel 6.3.6 中将两个模块拷贝出来，在小系统 init 中用 insmod 来加载它们，就可以实现键盘输入，如图 7 下方所示。

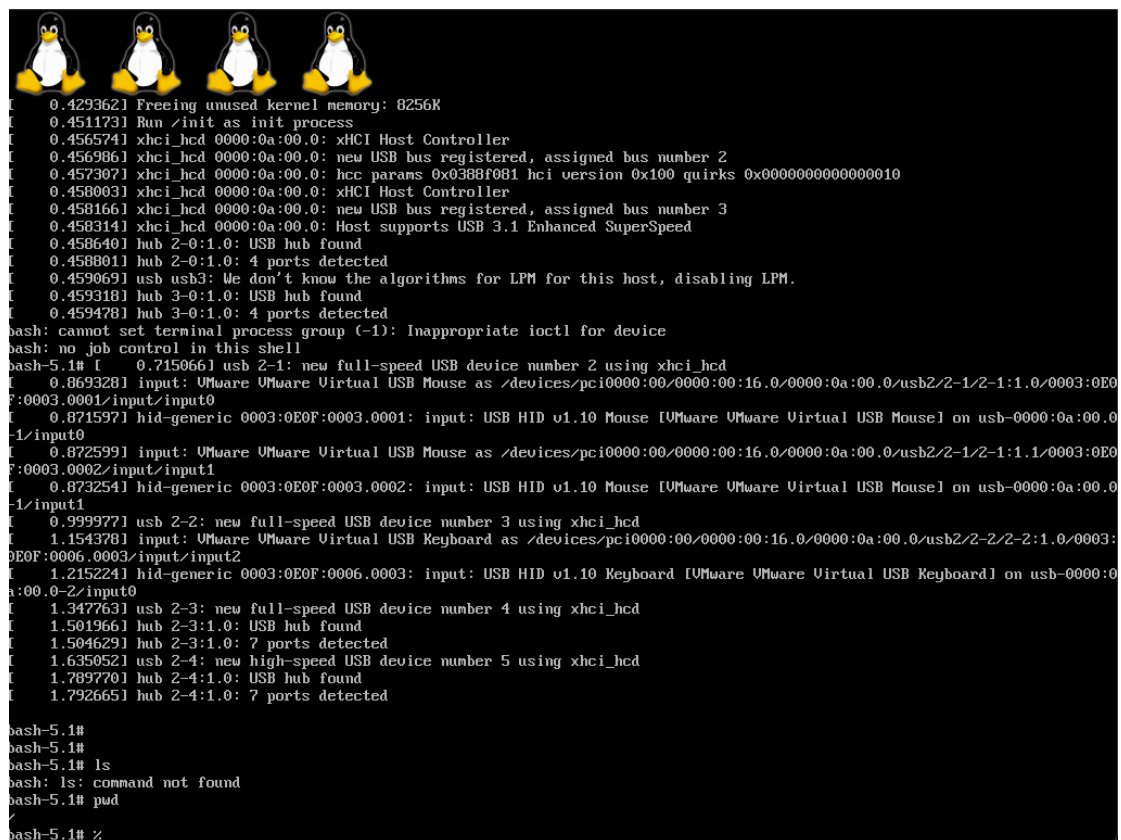


图7 v0.5 运行效果（暂无对 ls 等命令的支持）

完整代码见：

https://github.com/ShenMuyuan/custom_linux/blob/0f4ff481341fcd674824d605a5e5fd1ef7b28750/create_initrd_v0.5.sh，运行“sudo bash create_initrd_v0.5.sh”即可创建 initrd 并保存在/boot 目录。

3. v0.55: 手动加载驱动方式访问标准文件系统

我所使用的标准文件系统是基于 LVM 的。LVM 的架构如下：

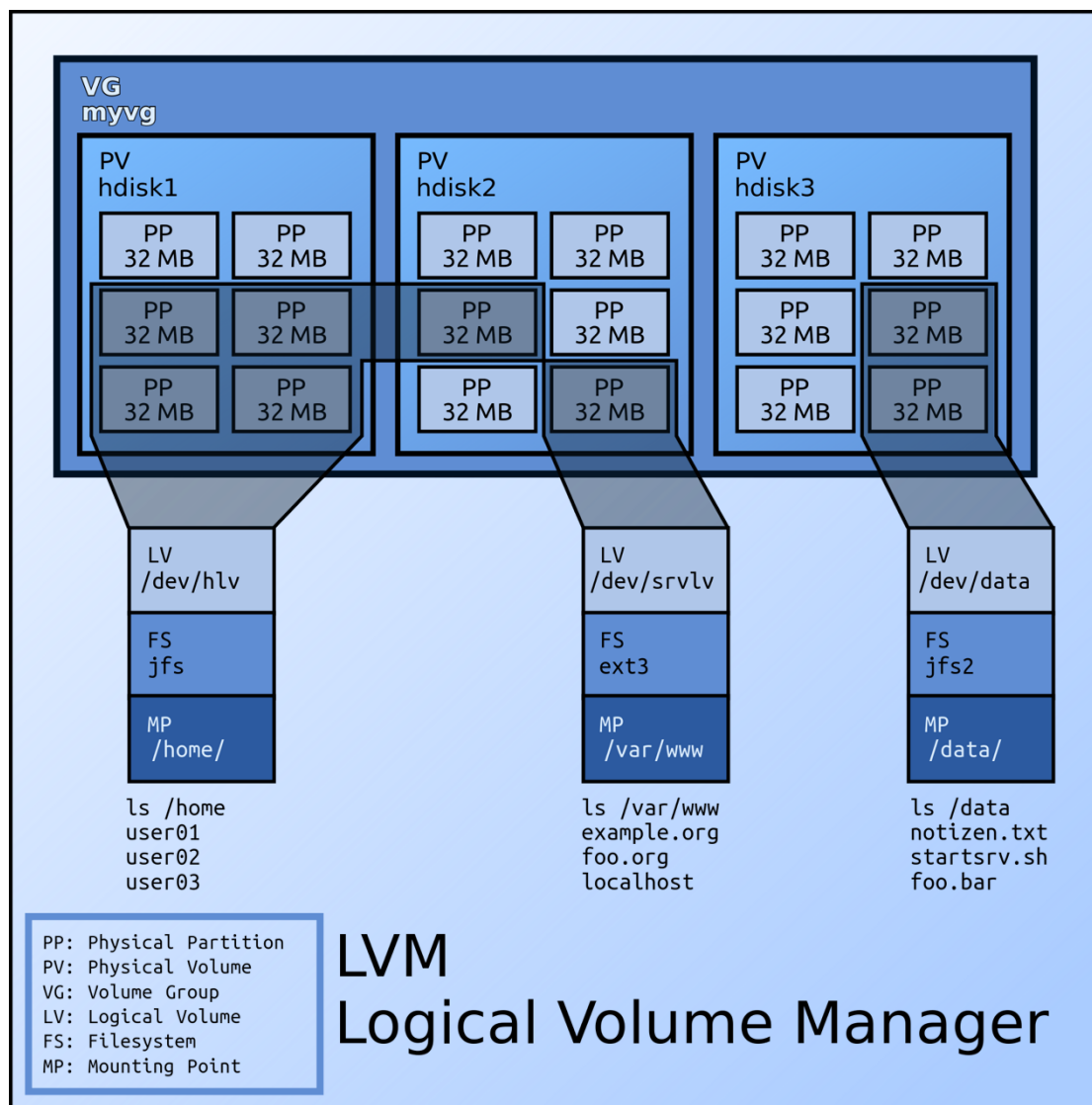


图8 LVM 的架构

从底层向上，分别是 PV（物理卷）、VG（物理卷组）、LV（逻辑卷），而任何文件系统都是基于 LV 的。通过 `lsblk` 命令可知，挂载到系统根目录的卷确实是一个 LV：

```

nvme0n1                259:0      0    40G  0 disk
├─nvme0n1p1             259:1      0   953M  0 part /boot/efi
├─nvme0n1p2             259:2      0    1.8G  0 part /boot
├─nvme0n1p3             259:3      0   37.3G  0 part
└─ubuntu--vg-ubuntu--lv 253:0      0   37.3G  0 lvm  /

```

图9 标准系统的 lsblk 输出

在小系统中需要实现的就是手动将这个 LV 挂载到指定的目录，并且能够访问其中的文件（如/home）。

我查阅了很多资料，发现其中核心的工具是“lvm”，它的第一个参数决定了用途。例如通过 lvm vgchange 可以激活一个 VG，lvm vgdisplay 可以显示 VG 的信息，对于 PV、LV 也是同样的使用方法。经过一些尝试（手动添加/dev/nvme*和/dev/dm-0 的设备节点，以及由/dev/ubuntu-vg/ubuntu-lv 指向 dm-0 的软链接），我终于能在小系统中将 LVM 激活，并且挂载到/mnt/my_root 目录，访问标准系统的文件，如下图所示。

```

bash-5.1# cd /mnt/my_root/
bash-5.1# ls -l
total 96
lrwxrwxrwx 1 0 0 7 Jan 9 07:57 bin -> usr/bin
drwxr-xr-x 2 0 0 4096 Jan 15 09:53 boot
drwxr-xr-x 4 0 0 4096 Jan 9 08:02 dev
drwxr-xr-x 106 0 0 4096 Jul 1 07:53 etc
drwxr-xr-x 3 0 0 4096 Jan 15 17:56 home
lrwxrwxrwx 1 0 0 7 Jan 9 07:57 lib -> usr/lib
drwx----- 2 0 0 16384 Jan 15 09:53 lost+found
drwxr-xr-x 2 0 0 4096 Jan 9 07:57 media
drwxr-xr-x 2 0 0 4096 Jan 9 07:57 mnt
drwxr-xr-x 2 0 0 4096 Jan 9 07:57 opt
drwxr-xr-x 2 0 0 4096 Apr 18 2022 proc
drwx----- 7 0 0 4096 Jul 2 08:50 root
drwxr-xr-x 14 0 0 4096 Jan 9 08:04 run
lrwxrwxrwx 1 0 0 8 Jan 9 07:57/sbin -> usr/sbin
drwxr-xr-x 6 0 0 4096 Jan 9 08:04 snap
drwxr-xr-x 2 0 0 4096 Jan 9 07:57 srv
drwxr-xr-x 2 0 0 4096 Apr 18 2022 sys
drwxrwxrwt 9 0 0 20480 Jul 2 08:56 tmp
drwxr-xr-x 11 0 0 4096 Jan 9 07:57 usr
drwxr-xr-x 13 0 0 4096 Jan 9 08:02 var

```

图10 v0.55 运行效果

完整代码见：

https://github.com/ShenMuyuan/custom_linux/blob/0f4ff481341fcd6

74824d605a5e5fd1ef7b28750/create_initrd_v0.55.sh, 运行“sudo bash create_initrd_v0.55.sh”即可创建 initrd 并保存在/boot 目录。

4. v0.6: 自动加载驱动方式访问标准文件系统

相比 v0.5 版本, v0.6 需要通过 udev 自动完成创建 nvme 设备节点、dm (Device Mapper) 设备节点和 LVM 配置的工作。首先为了降低实现的复杂性, 先将之前用到的 XHCI 键盘驱动编译进了内核 (所以后面做的时候, 不用考虑键盘了)。

在小系统中添加 udevadm 程序及其符号链接 /lib/systemd/systemd-udevd, 并把标准系统中所有规则, 即 /lib/udev/rules.d 加进去, 在 init 中通过以下命令初始化 udev:

```
# init udev
mount -t devtmpfs -o nosuid,mode=0755 udev /dev
SYSTEMD_LOG_LEVEL=info /lib/systemd/systemd-udevd --daemon --resolve-names=never
udevadm trigger --type=subsystems --action=add
udevadm trigger --type=devices --action=add
udevadm settle || true
```

图11 初始化 udev

对于 nvme, 标准版系统加载的相关模块如下图, 所以小系统若要支持 nvme, 则要将 ko 文件置于相同的路径。

```
• (base) smy@jammy:~/custom_linux$ lsmod | grep nvme
nvme                49152  3
nvme_core            139264  4 nvme
```

图12 标准系统用到的 nvme 相关模块

为了方便添加模块到相同路径, 编写脚本如下。考虑到与内核的兼容性, 该脚本用到模块的来源不是标准版系统, 而是编译好的 6.3.6 内核。

```

1  #!/bin/bash
2
3  # Usage: add_modules module1.ko module2.ko ...
4  # name only, not path
5
6  # Unlike commands and their dynamic linking libraries, kernel modules
7  # need to be extracted from kernel with the same version of testing kernel.
8  # Modify kernel compile options to get the modules.
9  my_kernel_dir=/home/smy/dev/kernel/linux-6.3.6/
10 my_kernel_version=6.3.6
11
12 add_modules() {
13     if [[ ! -d $my_kernel_dir ]]; then
14         echo "Error: wrong kernel directory $my_kernel_dir"
15         exit
16     fi
17     for mod in "$@"; do
18         full_path=$(find $my_kernel_dir -name "$mod".ko)
19         if [[ -z $full_path ]]; then
20             echo "Error: wrong module $mod"
21             exit
22         fi
23         rel_path=${full_path#$my_kernel_dir}
24         dest_path=lib/modules/$my_kernel_version/$rel_path
25         dest_root=${dest_path%/*}
26         if [[ ! -e $dest_path ]]; then
27             if [[ ! -d $dest_root ]]; then
28                 mkdir -p "$dest_root"
29             fi
30             cp "$full_path" "$dest_path"
31         fi
32     done
33 }
34

```

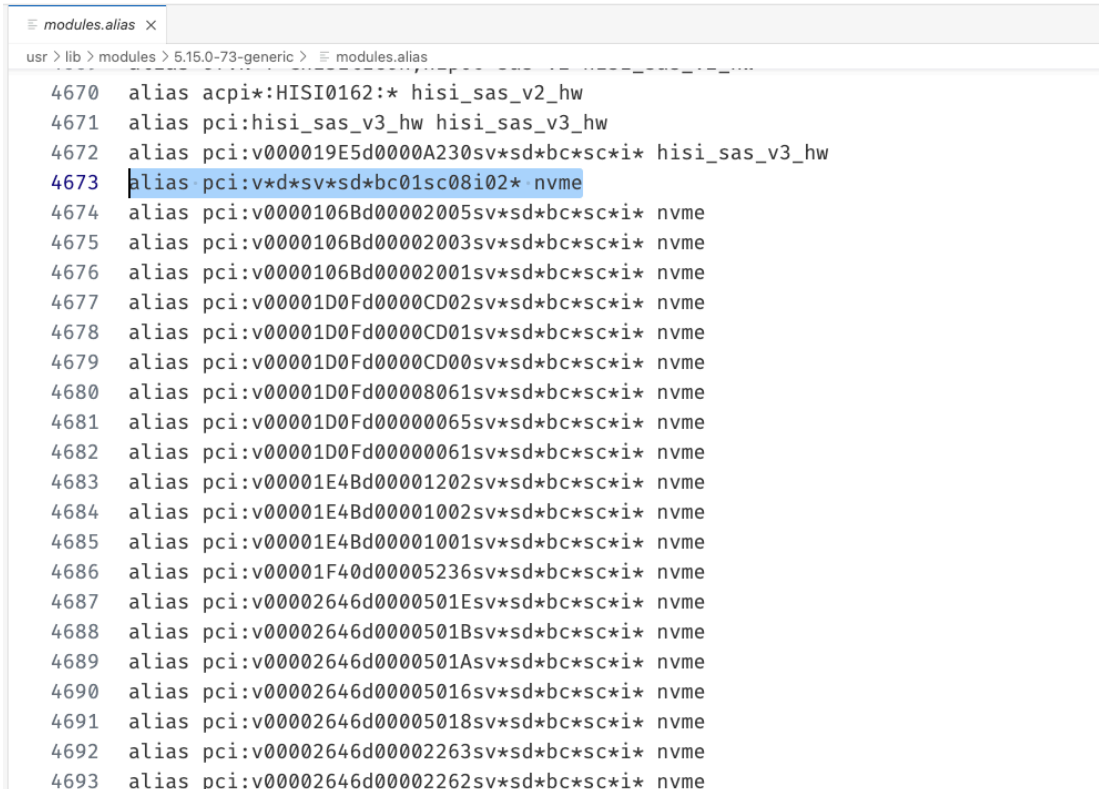
图13 向小系统拷贝模块的脚本

对于 device mapper, 查看 /lib/udev/rules.d/55-dm.rules, 发现其调用了 dmsetup 程序, 于是将 dmsetup 添加到小系统。

运行小系统后, 发现标准系统根目录并未成功挂载, 并通过 lsmod 得知 nvme 和 nvme-core 没有被加载。怀疑问题可能是 modprobe 找不到 nvme 和 nvme-core 模块。参考了 modprobe.d、modules.dep、depmod 的 manual 后, 认为问题由两个方面造成:

- (1) 缺少了包含 alias 信息的配置文件

对于 alias 的缺失，查看了标准系统提取出的 initrd 中的 `usr/lib/modules/5.15.0-73-generic/modules.alias` 文件，其结尾为 `nvme` 的各行即为 `nvme` 所需的 alias：



```
modules.alias x
usr > lib > modules > 5.15.0-73-generic > modules.alias
4670 alias acpi*:HISI0162:* hisi_sas_v2_hw
4671 alias pci:hisi_sas_v3_hw hisi_sas_v3_hw
4672 alias pci:v000019E5d0000A230sv*sd*bc*sc*i* hisi_sas_v3_hw
4673 alias pci:v*d*sv*sd*bc*sc*i* nvme
4674 alias pci:v0000106Bd00002005sv*sd*bc*sc*i* nvme
4675 alias pci:v0000106Bd00002003sv*sd*bc*sc*i* nvme
4676 alias pci:v0000106Bd00002001sv*sd*bc*sc*i* nvme
4677 alias pci:v00001D0Fd0000CD02sv*sd*bc*sc*i* nvme
4678 alias pci:v00001D0Fd0000CD01sv*sd*bc*sc*i* nvme
4679 alias pci:v00001D0Fd0000CD00sv*sd*bc*sc*i* nvme
4680 alias pci:v00001D0Fd00008061sv*sd*bc*sc*i* nvme
4681 alias pci:v00001D0Fd00000065sv*sd*bc*sc*i* nvme
4682 alias pci:v00001D0Fd00000061sv*sd*bc*sc*i* nvme
4683 alias pci:v00001E4Bd00001202sv*sd*bc*sc*i* nvme
4684 alias pci:v00001E4Bd00001002sv*sd*bc*sc*i* nvme
4685 alias pci:v00001E4Bd00001001sv*sd*bc*sc*i* nvme
4686 alias pci:v00001F40d00005236sv*sd*bc*sc*i* nvme
4687 alias pci:v00002646d0000501Esv*sd*bc*sc*i* nvme
4688 alias pci:v00002646d0000501Bsv*sd*bc*sc*i* nvme
4689 alias pci:v00002646d0000501Asv*sd*bc*sc*i* nvme
4690 alias pci:v00002646d00005016sv*sd*bc*sc*i* nvme
4691 alias pci:v00002646d00005018sv*sd*bc*sc*i* nvme
4692 alias pci:v00002646d00002263sv*sd*bc*sc*i* nvme
4693 alias pci:v00002646d00002262sv*sd*bc*sc*i* nvme
```

图14 `nvme` 所需的 alias 配置

将这些行加入小系统的 `etc/modprobe.d/aliases.conf`，实现的效果就是，`udev` 发现了在这些行指定的 `pci` 设备后，就会 `modprobe nvme`，而 `nvme` 依赖于 `nvme-core`，又会加载 `nvme-core`。这就带来了第二个问题：`nvme-core` 和 `nvme` 之间的依赖关系是怎么知道的？

(2) 缺少了包含依赖信息的配置文件

内核中各模块的依赖关系需要由 `depmod` 程序生成，保存为 `modules.dep.bin` 的二进制形式或者其等价文本版本 `modules.dep`，而 `depmod` 根据其文档的描述是无法指定搜索路径的，只能在系统路径

“/lib/modules/kernel 版本号”中搜索。为了对我编译出的内核生成 nvme 和 nvme-core 之间的依赖关系，我设计了以下方法：

```
cp -r linux-6.3.6 /lib/modules/6.3.6
depmod 6.3.6 /lib/modules/6.3.6/drivers/nvme/host/nvme.ko /lib/modules/6.3.6/drivers/md/dm-mod.ko /lib/modules/6.3.6/drivers/nvme/host/nvme-core.ko
# 然后, /lib/modules/6.3.6/modules.dep就有:
drivers/nvme/host/nvme.ko: drivers/nvme/host/nvme-core.ko
drivers/md/dm-mod.ko:
drivers/nvme/host/nvme-core.ko:
```

图15 生成模块依赖的方法

也就是说，先把编译出的内核整体拷贝到系统目录 /lib/modules/6.3.6，假装系统有 6.3.6 版本内核，再用 depmod 针对 6.3.6 内核生成依赖文件。然后在构建小系统的脚本里将 modules.dep.bin 拷贝到小系统里即可。

解决以上问题之后，udev 自动加载 nvme、device mapper 驱动以及自动挂载 LVM 就实现了。device mapper 所需的内核模块 BLK_DEV_DM 本来也应该像 nvme 一样提取出来、确定 alias 和依赖关系，但我为了简便编译进了内核。最终效果如下（与 v0.55 看上去是一样的）。



```
[ 0.407262] ata22: SATA link down (SStatus 0 SControl 300)
[ 0.407487] ata23: SATA link down (SStatus 0 SControl 300)
[ 0.407686] ata2.00: configured for UDMA/33
[ 0.407947] ata6: SATA link down (SStatus 0 SControl 300)
[ 0.408150] ata27: SATA link down (SStatus 0 SControl 300)
[ 0.408360] ata11: SATA link down (SStatus 0 SControl 300)
[ 0.408591] ata16: SATA link down (SStatus 0 SControl 300)
[ 0.408789] ata14: SATA link down (SStatus 0 SControl 300)
[ 0.409031] ata19: SATA link down (SStatus 0 SControl 300)
[ 0.409266] ata10: SATA link down (SStatus 0 SControl 300)
[ 0.419451] scsi 1:0:0:0: CD-ROM          NECUMMar VMware SATA CD01 1.00 PQ: 0 ANSI: 5
[ 0.422780] Freeing unused kernel memory: 8256K
[ 0.436540] Run /init as init process
Starting version 249.11-0ubuntu3.9
[ 0.455965] usb 2-1: new full-speed USB device number 2 using xhci_hcd
[ 0.558101] none nume0: pci function 0000:12:00.0
[ 0.563570] none nume0: 4/0/0 default/read/poll queues
[ 0.577445] none0n1: p1 p2 p3
[ 0.613039] input: VMware VMware Virtual USB Mouse as /devices/pci0000:00/0000:00:16.0/0000:0a:00.0/usb2/2-1/2-1:1.0/0003:0E0F:0003:0001/input/input0
[ 0.613784] hid-generic 0003:0E0F:0003:0001: input: USB HID v1.10 Mouse [VMware VMware Virtual USB Mouse] on usb-0000:0a:00.0-l/input0
[ 0.614169] input: VMware VMware Virtual USB Mouse as /devices/pci0000:00/0000:00:16.0/0000:0a:00.0/usb2/2-1/2-1:1.1/0003:0E0F:0003:0002/input/input1
[ 0.614405] hid-generic 0003:0E0F:0003:0002: input: USB HID v1.10 Mouse [VMware VMware Virtual USB Mouse] on usb-0000:0a:00.0-l/input1
[ 0.748032] usb 2-2: new full-speed USB device number 3 using xhci_hcd
[ 0.897744] input: VMware VMware Virtual USB Keyboard as /devices/pci0000:00/0000:00:16.0/0000:0a:00.0/usb2/2-2/2-2:1.0/0003:0E0F:0006:0003/input/input2
[ 0.956275] hid-generic 0003:0E0F:0006:0003: input: USB HID v1.10 Keyboard [VMware VMware Virtual USB Keyboard] on usb-0000:0a:00.0-2/input0
[ 1.084137] usb 2-3: new full-speed USB device number 4 using xhci_hcd
[ 1.233643] hub 2-3:1.0: USB hub found
[ 1.236139] hub 2-3:1.0: 7 ports detected
[ 1.364115] usb 2-4: new high-speed USB device number 5 using xhci_hcd
[ 1.513791] hub 2-4:1.0: USB hub found
[ 1.516426] hub 2-4:1.0: 7 ports detected
[ 1.670274] EXT4-fs (dm-0): mounted filesystem f23bccc4-819a-4107-b297-b0c60e702625 with ordered data mode. Quota mode: none.
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
bash-5.1# ls /mnt/my_root/
bin boot dev etc home lib lost+found media mnt opt proc root run/sbin snap srv sys tmp usr var
bash-5.1# %
```

图16 v0.6 实现效果

完整代码见：

[https://github.com/ShenMuyuan/custom_linux/blob/0f4ff481341fcd6](https://github.com/ShenMuyuan/custom_linux/blob/0f4ff481341fcd674824d605a5e5fd1ef7b28750/create_initrd_v0.6.sh)

[74824d605a5e5fd1ef7b28750/create_initrd_v0.6.sh](https://github.com/ShenMuyuan/custom_linux/blob/0f4ff481341fcd674824d605a5e5fd1ef7b28750/create_initrd_v0.6.sh)，运行“sudo bash

create_initrd_v0.6.sh”即可创建 initrd 并保存在/boot 目录。

v0.6 所用到的内核相比 defconfig，以下两项的设置从 M（编为模块）改成了*（编进内核）：

```
Symbol: USB_XHCI_PCI_RENESAS [=y]
Type : tristate
Defined at drivers/usb/host/Kconfig:46
Prompt: Support for additional Renesas xHCI controller with firmware
Depends on: USB_SUPPORT [=y] && USB [=y] && USB_XHCI_HCD [=y]
Location:
-> Device Drivers
-> USB support (USB_SUPPORT [=y])
-> xHCI HCD (USB 3.0) support (USB_XHCI_HCD [=y])
(1) -> Support for additional Renesas xHCI controller with firmware (USB_XHCI_PCI_RENESAS [=y])
```

图17 编进内核的 USB_XHCI_PCI_RENESAS

```

Symbol: BLK_DEV_DM [=y]
Type : tristate
Defined at drivers/md/Kconfig:206
Prompt: Device mapper support
Depends on: MD [=y] && (DAX [=n] || DAX [=n]=n)
Location:
-> Device Drivers
  -> Multiple devices driver support (RAID and LVM) (MD [=y])
(1)   -> Device mapper support (BLK_DEV_DM [=y])
Selects: BLOCK HOLDER_DEPRECATED [=y] && BLK_DEV_DM_BUILTIN [=y] && BLK_MQ_STACKING [=y]

```

图18 编进内核的 BLK_DEV_DM

5. v0.7: 创建临时配置使 systemd 管理 bash

首先是让 systemd 能够运行起来。运行本身是容易的，因为 systemd 要求 PID 为 1，在 init 脚本的末尾 exec /lib/systemd/systemd 即可。对于 systemd 的配置，经过我的不断尝试，在小系统里不报错的最小组合是：default.target、multi-user.target、basic.target 和 sysinit.target。

然后是创建一个服务，使 bash 退出后重新运行。参考了 systemd.service 文档中的示例以及其他服务配置，编写了以下服务：

```

touch lib/systemd/system/my-bash-management.service
tee lib/systemd/system/my-bash-management.service <<EOF
[Unit]
Description=Management for Bash in initramfs

[Service]
ExecStart=/bin/bash
Restart=always
RestartSec=0
StandardInput=tty
TTYPath=/dev/tty1

[Install]
WantedBy=default.target
EOF

```

图19 使 bash 不退出的服务

运行效果如下，bash 退出（按下 ctrl-D）后重新创建：

```
+GCRYPT +GNUTLS +OPENSSL +ACL +BLKID +CURL +ELFUTILS +FIDO2 +IDN2 -IDN +IPTC +KMOD +LIBCRYPTSETUP +LIBFDISK +PCRE2 -PWQUALITY -
P11KIT -QRENCODE +BZIP2 +LZ4 +XZ +ZLIB +ZSTD -XZBCOMMON +UTMP +SYSVINIT default-hierarchy=unified)
[ 1.687300] systemd[1]: Detected virtualization vmware.
[ 1.687577] systemd[1]: Detected architecture arm64.
[ 1.687849] systemd[1]: Detected first boot.

Welcome to Linux!

[ 1.688989] systemd[1]: No hostname configured, using default hostname.
[ 1.689319] systemd[1]: Hostname set to <localhost>.
[ 1.689632] systemd[1]: Initializing machine ID from random generator.
[ 1.711367] systemd[1]: Populated /etc with preset unit settings.
[ 1.730183] systemd[1]: Queued start job for default target Graphical Interface.
[ 1.731137] systemd[1]: Reached target System Initialization.
[ OK ] Reached target System Initialization.
[ 1.731659] systemd[1]: Reached target Basic System.
[ OK ] Reached target Basic System.
[ 1.732046] systemd[1]: System is tainted: var-run-bad
[ 1.732225] systemd[1]: Reached target Multi-User System.
[ OK ] Reached target Multi-User System.
[ 1.733177] systemd[1]: Started Management for Bash in initramfs.
[ OK ] Started Management for Bash in initramfs.
[ 1.733636] systemd[1]: Reached target Graphical Interface.
[ OK ] Reached target Graphical Interface.
[ 1.747156] systemd[1]: Startup finished in 1.669s (kernel) + 74ms (userspace) = 1.744s.
bash-5.1# ls
bin dev etc init lib lib64 mnt proc root run sbin sys tmp usr var
bash-5.1# exit
[ 7.883717] systemd[1]: my-bash-management.service: Deactivated successfully.
[ 7.885252] systemd[1]: my-bash-management.service: Scheduled restart job, restart counter is at 1.
[ 7.885961] systemd[1]: Stopped Management for Bash in initramfs.
[ 7.923401] systemd[1]: Started Management for Bash in initramfs.
bash-5.1# exit
[ 8.301759] systemd[1]: my-bash-management.service: Deactivated successfully.
[ 8.303915] systemd[1]: my-bash-management.service: Scheduled restart job, restart counter is at 2.
[ 8.304563] systemd[1]: Stopped Management for Bash in initramfs.
[ 8.339519] systemd[1]: Started Management for Bash in initramfs.
bash-5.1# exit
[ 8.706319] systemd[1]: my-bash-management.service: Deactivated successfully.
[ 8.708124] systemd[1]: my-bash-management.service: Scheduled restart job, restart counter is at 3.
[ 8.708937] systemd[1]: Stopped Management for Bash in initramfs.
[ 8.747827] systemd[1]: Started Management for Bash in initramfs.
bash-5.1# %
```

图20 v0.7 实现效果

完整代码见：

https://github.com/ShenMuyuan/custom_linux/blob/0f4ff481341fcd6

[74824d605a5e5fd1ef7b28750/create_initrd_v0.7.sh](https://github.com/ShenMuyuan/custom_linux/blob/0f4ff481341fcd674824d605a5e5fd1ef7b28750/create_initrd_v0.7.sh)，运行“sudo bash

create_initrd_v0.7.sh”即可创建 initrd 并保存在/boot 目录。

6. v0.8：完整 systemd 运行环境+login

将 systemd 的所有配置/lib/systemd/*均拷贝至小系统，并把 v0.7 的临时配置删除，即可实现完整的 systemd 运行环境。

为了以正常途径得到 shell，必须要 login。为了实现 login，我经历了以下尝试：

- (1) 在 init 中创建用户，在 init 中 exec login

这时，发现会报 PAM failure 的错误。查阅 pam 的文档，发现需要添加了 pam 的动态库和配置文件。

(2) 添加了 PAM 的依赖，并采用完整 systemd 流程

这时，在 login 的提示符后输入了 init 中创建的用户名，就会报 Login incorrect 错误：

A terminal window with a black background and white text. It shows four lines of 'localhost login:' prompts. The fourth line is followed by the username 'diangroup'. Below this, the text 'Login incorrect' is displayed, followed by another 'localhost login:' prompt.

```
localhost login:
localhost login:
localhost login:
localhost login: diangroup

Login incorrect
localhost login:
```

图21 Login incorrect 错误

仔细检查发现，pam 的动态库还依赖其他的动态库，这部分之前忽略了。为了自动添加“动态库依赖的动态库”，我编写了以下脚本：


```

1  #!/bin/bash
2
3  # Usage: add_libraries /path/to/lib1 /path/to/lib2 ...
4  # full path only
5
6  . utils/add_file.sh
7
8  add_libraries() {
9      for lib in "$@"; do
10         if [[ $lib =~ ^\ / ]]; then # lib is full path
11             if [[ -e "$lib" ]]; then
12                 path=$lib
13             else
14                 echo "Error: wrong library $lib"
15                 exit
16             fi
17         else # lib is a name
18             echo "Error: wrong library $lib"
19             exit
20         fi
21         echo "Adding library: $path"
22         add_file "$path"
23
24         for line in $(ldd "$path"); do # split words by space and newline
25             if [[ $line =~ ^\ / ]]; then
26                 echo "Adding dependency library: $line"
27                 add_file "$line"
28             fi
29         done
30     done
31 }

```

图22 添加动态库及其依赖的脚本

构建小系统过程中，添加 pam 所需的动态库：

```

add_libraries /usr/lib/aarch64-linux-gnu/security/* /usr/lib/aarch64-linux-gnu/nss/*.so /usr/lib/aarch64-linux-gnu/libnss* /usr/lib/aarch64-linux-gnu/libfreebl*
add_libraries /lib/aarch64-linux-gnu/libz.so.1
add_libraries /lib/aarch64-linux-gnu/libnsl.so.2

```

图23 添加 PAM 所需动态库及其依赖

(3) 完善了 PAM 后

此时，经历了 systemd 启动过程后，可以实现以 diangroup 登陆

（此时不在启动，所以就没有企鹅图标了）：

```
localhost login: diangroup
Password:
-bash-5.1$
-bash-5.1$
-bash-5.1$ whoami
diangroup
-bash-5.1$
```

图24 diangroup 登陆

注：在后续版本中有少量改进，例如修改主机名为 SMY-Linux，给 diangroup sudo 权限等。

完整代码见：

https://github.com/ShenMuyuan/custom_linux/blob/0f4ff481341fcd674824d605a5e5fd1ef7b28750/create_initrd_v0.8.sh，运行“sudo bash create_initrd_v0.8.sh”即可创建 initrd 并保存在/boot 目录。

7. v0.9：复建标准系统的启动流程（支持网络、ssh、ssh server）

首先是配置网络。在标准系统查看 ifconfig、route -n，得到 IP 地址和网关信息。我标准系统虚拟机分配到的 IP 地址是 172.16.49.130，网关是 172.16.49.2，于是在 init 中加入以下配置：

```
# for ip
ip link set enp2s0 up
ip addr add 172.16.49.130/24 dev enp2s0
route add default gw 172.16.49.2
```

图25 ip 和路由配置

注意上面的网卡名称 enp2s0，和标准系统中的名称（我的是 ens160）是不一样的，需要进入到小系统中，通过 ip addr 等命令查看。

尝试 ping，但遇到了以下问题。

```
-bash-5.1$ ping 172.16.49.1
ping: socket: Address family not supported by protocol
```

图26 ping 遇到的问题

查阅资料发现是 diangroup 权限不足，于是在 init 里面创建 diangroup 账户的位置赋予其 sudo 权限，如下图。

```
mkdir -m 0755 /home
useradd -m -d /home/diangroup -s /bin/bash diangroup
echo diangroup:diangroup | chpasswd
usermod -aG sudo diangroup
touch /home/diangroup/.profile
echo "export PS1='\u@\h:\W \$ '" >> /home/diangroup/.profile
echo "export PATH=\"/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:${PATH}\"" >> /home/diangroup/.profile
touch /etc/hostname
echo "SMY-Linux" >> /etc/hostname
echo "127.0.0.1 SMY-Linux" >> /etc/hosts
chown root:root /usr/bin/sudo 66 chmod 4755 /usr/bin/sudo
```

图27 创建 diangroup 用户的相关配置

接着，局域网（172.16.49.2）、外网（8.8.8.8）都可以 ping 成功。

```
SMY-Linux login: diangroup
Password:
diangroup@SMY-Linux:~$
diangroup@SMY-Linux:~$
diangroup@SMY-Linux:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:5a:3a:18 brd ff:ff:ff:ff:ff:ff
    inet 172.16.49.130/24 scope global enp2s0
        valid_lft forever preferred_lft forever
diangroup@SMY-Linux:~$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 00:0c:29:5a:3a:18 brd ff:ff:ff:ff:ff:ff
diangroup@SMY-Linux:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.49.2    0.0.0.0         UG    0      0        0 enp2s0
172.16.49.0     0.0.0.0        255.255.255.0   U     0      0        0 enp2s0
diangroup@SMY-Linux:~$ ping 172.16.49.2
ping: socket: Address family not supported by protocol
diangroup@SMY-Linux:~$ sudo ping 172.16.49.2
[sudo] password for diangroup:
PING 172.16.49.2 (172.16.49.2) 56(84) bytes of data.
64 bytes from 172.16.49.2: icmp_seq=1 ttl=128 time=0.395 ms
64 bytes from 172.16.49.2: icmp_seq=2 ttl=128 time=0.492 ms
64 bytes from 172.16.49.2: icmp_seq=3 ttl=128 time=0.573 ms
^C
--- 172.16.49.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2037ms
rtt min/avg/max/mdev = 0.395/0.486/0.573/0.072 ms
diangroup@SMY-Linux:~$ sudo ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=128 time=57.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=82.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=170 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=128 time=218 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 57.157/131.778/217.608/64.992 ms
diangroup@SMY-Linux:~$ %
```

图28 成功 ping 通局域网和外网

加入 ssh 命令，尝试 ssh 的登陆。下图为成功登陆团队某内网服务器。

```
-bash-5.1$ ssh shenmuyuan@192.168.0.89
The authenticity of host '192.168.0.89 (192.168.0.89)' can't be established.
ED25519 key fingerprint is SHA256:d+GReViHXBhpv0p5c+XT0/BiIdNLj249mmmODGwLjSA.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.89' (ED25519) to the list of known hosts.
shenmuyuan@192.168.0.89's password:
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-1009-realtime x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sat Aug 12 09:44:38 PM CST 2023

System load:  0.0                      Processes:            1590
Usage of /:   84.6% of 434.78GB        Users logged in:     2
Memory usage: 27%                     IPv4 address for docker0: 172.17.0.1
Swap usage:   0%                      IPv4 address for eno1: 192.168.0.89
Temperature: 53.0 C

=> There are 32 zombie processes.

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

116 updates can be applied immediately.
5 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

*** System restart required ***
Last login: Mon Aug  7 17:57:48 2023 from 192.168.0.3
(base) shenmuyuan@ubuntu2204:~$
```

图29 ssh 登陆团队内网服务器

相比 ssh 而言，ssh server (sshd) 实现则不太容易，查阅资料发现 sshd 用到的有其自身、/etc/ssh/sshd_config 配置文件以及其他配置文件和命令。我加入的有：

```
# support sshd
add_commands sshd pidof tput expr grep
add_file /etc/ssh/sshd_config
add_file /etc/init.d/ssh
add_file /etc/default/ssh
add_file /lib/lsb/init-functions
add_file /usr/sbin/service
add_commands sh basename journalctl
add_commands dbus-daemon
add_file /usr/share/dbus-1/{system.conf,system-services/*,system.d/*,session.conf}
add_commands cron su
add_file /etc/crontab
add_commands ssh-keygen
```

图30 sshd 的部分依赖

加入这些依赖之后，在小系统启动后，拿本地去连接没有成功，显示连接被对端关闭了。查看 journal 发现原因可能是 hostkey 缺失了，如图。

```
root@SMY-Linux:diangroup $ journalctl -xeu ssh.service
Aug 13 13:07:16 SMY-Linux systemd[1]: Starting OpenBSD Secure Shell server...
Aug 13 13:07:16 SMY-Linux sshd[6021]: sshd: no hostkeys available -- exiting.
Aug 13 13:07:16 SMY-Linux systemd[1]: ssh.service: Control process exited, code=exited, status=1/FAILURE
Aug 13 13:07:16 SMY-Linux systemd[1]: ssh.service: Failed with result 'exit-code'.
Aug 13 13:07:16 SMY-Linux systemd[1]: Failed to start OpenBSD Secure Shell server.
Aug 13 13:07:16 SMY-Linux systemd[1]: ssh.service: Scheduled restart job, restart counter is at 1.
Aug 13 13:07:16 SMY-Linux systemd[1]: Stopped OpenBSD Secure Shell server.
Aug 13 13:07:16 SMY-Linux systemd[1]: Starting OpenBSD Secure Shell server...
Aug 13 13:07:16 SMY-Linux sshd[6051]: sshd: no hostkeys available -- exiting.
Aug 13 13:07:16 SMY-Linux systemd[1]: ssh.service: Control process exited, code=exited, status=1/FAILURE
Aug 13 13:07:16 SMY-Linux systemd[1]: ssh.service: Failed with result 'exit-code'.
Aug 13 13:07:16 SMY-Linux systemd[1]: Failed to start OpenBSD Secure Shell server.
Aug 13 13:07:16 SMY-Linux systemd[1]: ssh.service: Scheduled restart job, restart counter is at 2.
Aug 13 13:07:16 SMY-Linux systemd[1]: Stopped OpenBSD Secure Shell server.
Aug 13 13:07:16 SMY-Linux systemd[1]: Starting OpenBSD Secure Shell server...
Aug 13 13:07:16 SMY-Linux sshd[6071]: sshd: no hostkeys available -- exiting.
Aug 13 13:07:16 SMY-Linux systemd[1]: ssh.service: Control process exited, code=exited, status=1/FAILURE
Aug 13 13:07:16 SMY-Linux systemd[1]: ssh.service: Failed with result 'exit-code'.
Aug 13 13:07:16 SMY-Linux systemd[1]: Failed to start OpenBSD Secure Shell server.
Aug 13 13:07:17 SMY-Linux systemd[1]: ssh.service: Scheduled restart job, restart counter is at 3.
Aug 13 13:07:17 SMY-Linux systemd[1]: Stopped OpenBSD Secure Shell server.
Aug 13 13:07:17 SMY-Linux systemd[1]: Starting OpenBSD Secure Shell server...
Aug 13 13:07:17 SMY-Linux sshd[6091]: sshd: no hostkeys available -- exiting.
Aug 13 13:07:17 SMY-Linux systemd[1]: ssh.service: Control process exited, code=exited, status=1/FAILURE
Aug 13 13:07:17 SMY-Linux systemd[1]: ssh.service: Failed with result 'exit-code'.
Aug 13 13:07:17 SMY-Linux systemd[1]: Failed to start OpenBSD Secure Shell server.
Aug 13 13:07:17 SMY-Linux systemd[1]: ssh.service: Scheduled restart job, restart counter is at 4.
Aug 13 13:07:17 SMY-Linux systemd[1]: Stopped OpenBSD Secure Shell server.
Aug 13 13:07:17 SMY-Linux systemd[1]: Starting OpenBSD Secure Shell server...
Aug 13 13:07:17 SMY-Linux sshd[6111]: sshd: no hostkeys available -- exiting.
Aug 13 13:07:17 SMY-Linux systemd[1]: ssh.service: Control process exited, code=exited, status=1/FAILURE
Aug 13 13:07:17 SMY-Linux systemd[1]: ssh.service: Failed with result 'exit-code'.
Aug 13 13:07:17 SMY-Linux systemd[1]: Failed to start OpenBSD Secure Shell server.
Aug 13 13:07:17 SMY-Linux systemd[1]: ssh.service: Scheduled restart job, restart counter is at 5.
Aug 13 13:07:17 SMY-Linux systemd[1]: Stopped OpenBSD Secure Shell server.
Aug 13 13:07:17 SMY-Linux systemd[1]: ssh.service: Start request repeated too quickly.
Aug 13 13:07:17 SMY-Linux systemd[1]: ssh.service: Failed with result 'exit-code'.
Aug 13 13:07:17 SMY-Linux systemd[1]: Failed to start OpenBSD Secure Shell server.
```

图31 查看 ssh.service 的 journal

在 init 中生成所需 4 组加密方式的 hostkey。

```
# for ssh key
ssh-keygen -q -N "" -t dsa -f /etc/ssh/ssh_host_dsa_key
ssh-keygen -q -N "" -t rsa -b 4096 -f /etc/ssh/ssh_host_rsa_key
ssh-keygen -q -N "" -t ecdsa -f /etc/ssh/ssh_host_ecdsa_key
ssh-keygen -q -N "" -t ed25519 -f /etc/ssh/ssh_host_ed25519_key
```

图32 生成 hostkey

此时可以正常 ssh 连接小系统（由于 ip 地址和标准系统一样，ssh 会因为 key 的更改报错，所以 ssh 前要把 known_hosts 里面的相应行删除）。

```
> ssh diangroup@172.16.49.130
The authenticity of host '172.16.49.130 (172.16.49.130)' can't be established.
ED25519 key fingerprint is SHA256:GTGAQatA7vZxybd6Qg6t2X+EdYdi7wHzeffnilQ8fJUJ.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.16.49.130' (ED25519) to the list of known hosts.
diangroup@172.16.49.130's password:
diangroup@SMY-Linux:~ $
```

图33 ssh 以 diangroup 用户登陆小系统

完整代码见：

https://github.com/ShenMuyuan/custom_linux/blob/0f4ff481341fcd674824d605a5e5fd1ef7b28750/create_initrd_v0.9.sh，运行“sudo bash create_initrd_v0.9.sh”即可创建 initrd 并保存在/boot 目录。

功能目标实现

1. 通过 U 盘或 PXE 网络启动加载 kernel 和 img 启动进行验证

首先，基于 vmlinuz 和 initrd 创建可用于引导的 ISO 镜像。我是用的工具是 grub-mkrescue（创建急救镜像），创建过程需要以下的目录结构：

```

root@jammy:~# tree my_iso/
my_iso/
├── boot
│   ├── grub
│   │   └── grub.cfg
│   ├── smy_initrd.gz
│   └── smy_vmlinuz.gz
2 directories, 3 files

```

图34 创建镜像所需目录结构

其中，grub.cfg 包含 grub 目录的项目：

```

set default=0
set timeout=10
menuentry 'SMYOS' --class os {
    insmod gzio
    insmod part_msdos
    linux /boot/smy_vmlinuz.gz
    initrd /boot/smy_initrd.gz
}

```

图35 镜像的 grub 配置

创建急救镜像：

```

root@jammy:~# grub-mkrescue -o smyos.iso my_iso/
xorriso 1.5.4 : RockRidge filesystem manipulator, libburnia project.

Drive current: -outdev 'stdio:smyos.iso'
Media current: stdio file, overwriteable
Media status : is blank
Media summary: 0 sessions, 0 data blocks, 0 data, 5983m free
Added to ISO image: directory '/='/tmp/grub.4oIQOp'
xorriso : UPDATE :    234 files added in 1 seconds
Added to ISO image: directory '/='/root/my_iso'
xorriso : UPDATE :    239 files added in 1 seconds
ISO image produced: 20916 sectors
Written to medium : 20916 sectors at LBA 0
Writing to 'stdio:smyos.iso' completed successfully.

```

图36 镜像的 grub 配置

为测试该镜像，修改 VMware 的启动顺序为 DVD 在前，连接 iso 文件，可以看到 grub 菜单：



图37 急救镜像的 grub 菜单

进入 SMYOS，可以验证 v0.9 的全部功能。

接着，使用 Etcher 工具，基于 smyos.iso 制作急救 U 盘。遗憾的是，制作完成的 U 盘并未被电脑识别，只能在启动磁盘选单上看到电脑的硬盘。查询相关资料没有找到解决方法，怀疑是 MacBook 在硬件方面的封闭性导致的。



图38 U 盘插上却不被电脑识别

2. 支持多用户登录 (console 界面和 ssh 网络方式)

已在 v0.8 和 v0.9 展示过。

3. 系统支持通过 ssh 方式访问其他机器

已在 v0.8 展示过。

4. 可挂载 U 盘

由于内核中已有 xhci 驱动 (支持 USB) 以及 vfat 驱动 (支持 FAT32), 插上 U 盘就会提示信息, 并在 /dev 里面看到 sda1 (系统所在硬盘是 nvme 所以不在 sd 开头的里面)。

```
root@SMY-Linux:diangroup $ ls -l /dev | grep sd
root@SMY-Linux:diangroup $ [ 67.590876] usb 3-1: new SuperSpeed USB device number 2 using xhci_hcd
[ 67.613778] usb-storage 3-1:1.0: USB Mass Storage device detected
[ 67.615291] scsi host30: usb-storage 3-1:1.0
[ 68.645275] scsi 30:0:0:0: Direct-Access Kingston DataTraveler 3.0 PQ: 0 ANSI: 6
[ 68.649109] sd 30:0:0:0: [sda] 60437492 512-byte logical blocks: (30.9 GB/28.8 GiB)
[ 68.652575] sd 30:0:0:0: [sda] Write Protect is off
[ 68.653587] sd 30:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't support DPO or FUA
[ 68.675185] sda: sda1
[ 68.676558] sd 30:0:0:0: [sda] Attached SCSI removable disk

root@SMY-Linux:diangroup $
root@SMY-Linux:diangroup $ ls -l /dev | grep sd
brw-rw---- 1 root disk 8, 0 Aug 13 15:16 sda
brw-rw---- 1 root disk 8, 1 Aug 13 15:16 sda1
```

图39 识别 U 盘

可以以 vfat 格式挂载, 并看到其中的内容 (内核没有将中文支持编译进去, 所以中文显示成问号):

```
root@SMY-Linux:diangroup $ mount -t vfat /dev/sda1 /mnt/my_usb/ -o rw
root@SMY-Linux:diangroup $ ls /mnt/my_usb/
'16????????(???)?.mov'      '???????'      '??_1.mov'
'2019???????'              '??????13.mov' 'AE CC 2018?????.zip'
'???'                      '???????'      'AI?????.MOV'
'???'                      '????????(???)?.mov' 'Adobe Premiere Pro Auto-Save'
'??1 ??????????.doc'      '?????????'      'Adobe Premiere Pro CC 2019.rar'
'??3 ??????????.doc'      '????????? - ????.mp3' 'Adobe Premiere Pro_2020_14.0.0.571_SP_20191023.rar'
'??4 ??????????????????.doc' '?????????.mov'    BOOTEX.LOG
'???'                      '????????_1lpy.prproj' 'Intel???'
'???.mp3'                  '????_1lpy.prproj' MUI_9527.MOV
'???.mpeg'                 '????_???.pdf'     MUI_9621.MOV
'?????.mp4'                '????finalfinal.mp4' MUI_9622.MOV
'?????'                    '???_1.mp4'        'System Volume Information'
'?????'                    '???_final.mp4'    logo.dd99a9865177.gif
'?????2019?????????????_201901191045 - ???.docx' '???_final.mp4'    '~$3 ??????????.doc'
```

图40 挂载 U 盘, 查看内容

5. 可访问机器上的 windows 分区 (ntfs-3g fs 支持)

由于我的电脑为 macOS, 采用 APFS 文件系统, 且启动时 U 盘不被识别, 所以这项功能是不支持的。假如日后苹果放宽了对引导

设备的支持，可以使用开源的 apfs-fuse 驱动来查看 APFS 文件系统。